Autonomous Navigation through Reinforcement Learning in a Non-simulated Environment

Author:

Maximilian Müller

Supervisor:

Kamal Nasrollahi

Aalborg University

Abstract

The focus of this project is to compare different reinforcement learning techniques in regards to how they can be used for autonomous computer vision based navigation. This is done by reviewing similar projects which solve related problems that utilise reinforcement learning. The primary techniques that are considered are Q-Learning and Neuro evolution of augmented topologies. This is brought into practice by the implementation of a prototype which consists of a robot that autonomously navigates in a real environment. It is shown how the parameters for the reinforcement learning can be found with simulated annealing. Furthermore is the importance of using an exploration function shown.

Contents

1 Introduction

The topic of the presented project is autonomous navigation for mobile driving robots. This project was executed subsequent to a project in which a form of optical flow based navigation was used. This optical flow was obtained by the use of so called Reichardt detectors [Basch et al., 2010].

Reichardt detectors are used to mimic the behavior of the neuron of honeybees which are used to perceive motion [Cope A.J. and D, 2013]. It was shown in this project that a small robot with a differential drive and a single forward facing camera, can centre itself in a corridor. This is illustrated in Figure 1

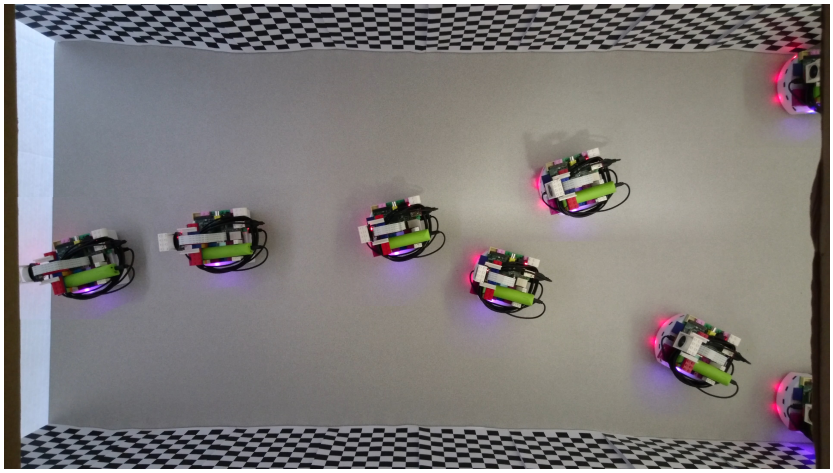

Figure 1: *An illustration of how the robot in the first project was able to centre itself in a corridor. For the illustration images of the two runs were merged.*

The next natural step is to look at how other navigation tasks can be solved.

1.1 Motivation

The task of navigation is the task to find a suitable path from the current position of the system, to a desired position. In the prior work, the desired position was in front of the robot and the suitable meant not driving into the walls of the corridor.

One can imagine scenarios that would be more complex and would have sharper success criterias than this. For instance could a specific goal position be required, that would force the robot to stop as well [Bischoff et al., 2013]. Furthermore different obstacles could block the way to the goal [Cherni et al., 2015], [Tan and Cai, 2015], the driving conditions could change or the robot

itself could have changing attributes (e.g. slower acceleration due to battery exhaustion).

Another direction in which the project could be taken is to choose a robot with a different kind of locomotion. This could for instance be a crawling robot [Tesen et al., 2013], [Jatsun et al., 2014], [Nagai et al., 2015], [Dresscher et al., 2014], a flying robot [Vieira et al., 2016], [Lee and Kim, 2013], [Dotenco et al., 2015], [Suárez et al., 2014] or a swimming robot [Katzschmann et al., 2016], [Manfredi et al., 2013], [Yuan et al., 2016].

It seems like there is a wide variety of autonomous robots to choose from, which could solve a wide variety of problems in a lot of different scenarios. This clearly shows, even though a specific solution to a specific navigation problem might be useful, it would be desirable to find a general solution that is applicable in a variety of scenarios and is not specific to a robot platform.

While autonomous acting robots in the past were restricted to research labs, they have successively moved onto the private market and into mass prduction (See Figure 2).

Figure 2: *The drone Phantom 4 from DJI* [1]*, Google's self driving car* [2] *and vacuum cleaner Dyson 360 Eye*[3]

Especially the private drone market has shown an impressive growth over the past few years and analysts forecast it to grow massively in the future [McNabb, 2016].

There are a number of professional application in which autonomous moving robots could be used. Drones are being used in the film industry to capture footage from angles that were inaccessible before. They are also useful for surveillance, rescue missions and other applications where a different camera angle is an advantage [Bonin-Font et al., 2008].

Autonomous vehicles are especially useful for applications that would endanger humans otherwise due to a hostile environment (e.g. Planets [Chatila et al., 1995], volcanic crater [Astuti et al., 2009], etc).

[3]Source: `https://www.google.com/selfdrivingcar/images/paint/artworks/mountain-view/dufayet.jpg`

[3]Source: `http://www.dyson.co.jp/medialibrary/Group/ShopContent/Hero/Products/Japan/Robot/Hero_Robot_360eye_NickelBlue.ashx?w=378&bc=ffffff`

[3]Source: `https://www.google.com/selfdrivingcar/images/paint/artworks/mountain-view/dufayet.jpg`

Furthermore could one imagine that the autonomy can take faster and more rational decisions than a human. This is being explored by car manufacturers that are developing autonomous driving (e.g. Audi, Tesla etc.).

One can choose among a wide variety of sensors, when designing a way of letting the autonomous driving system perceive the world. A lot of the products for which autonomous navigating could be useful are already equipped with a camera or could easily be equipped with one due to its weight and low cost. This is especially true, when comparing a camera to a LIDAR sensor, which is used for navigation in Google's self driving car (See Figure 2). The light weight of a camera even makes the application on Unmanned Aerial Vehicles (UAV) possible [Bonin-Font et al., 2008].

Computer vision has among other things successfully been used applied to the tasks of self-localisation, automatic map construction, autonomous navigation, path following, inspection, monitoring or risky situation detection [Bonin-Font et al., 2008] - which are all problems in navigation.

Quite recently small embedded system like the Myriad 2 (See Figure 3) have become powerful enough to perform even complex parallel computation needed for computer vision while consuming little power [Barry et al., 2015]. Such an embedded platform can therefore be used to perform the computation needed for navigation on the vehicle itself, rather on a remote server.

The light weight of the camera combined with the low power consumption of the embedded system makes it possible to perform autonomous navigation even on an UAV, while running the entire system on a small battery that can be carried by the UAV.

Figure 3: *The Myriad 2 reference board from Movidius* [4]

Alongside with the growth in the private drone market, it has come to several drone related accidents [Democrat and Chronicle, 2016]. This clearly shows the need for smarter autonomous navigation which among other things justifies the research in this area.

Also quite recently the first fatal crash was caused by an autopilot system from Tesla, which did not detect a perpendicular crossing tractor against the bright sky [Tesla, 2016]. This indicates that even though autonomous driving has come a long way, that there is still research to be done.

1.2 Initial Problem Statement

In this project it is seeked to improve autonomous navigation for mobile systems.

The sought solution should be applicable in a variety of scenarios and not specific to a robot platform. Furthermore would it be desirable if the solution

[4]Source: http://www.movidius.com/solutions/vision-processing-unit

could run on an embedded platform and would use a camera to obtain a representation of the environment.

The main question that will be answered in this work is:

How to design a computer vision based autonomous navigation system that can be used for a variety of problems and is not specific to the platform it is being used on?

1.3 Structure

The rest of this project is structured in the following way: The different related project are described in section 2. This is followed by an analysis in section 3. After the analysis, the needed techniques are explained in section 4. To access the different qualities of techniques several prototypes were implemented and described in section 5 alongside with the practical challenges when using computer vision in reinforcement learning. This is followed by section 6, were the prototypes optimal settings are determined and their performance is evaluated. The project is concluded upon in section 7.

2 Related Work

In order to answer the initial problem statement one first needs to understand what navigation is in the context of mobile robot navigation. Cherni et al. [2015] summarise navigation into answering the following three questions:

- Where are we?
- Where are we going?
- How to get there?

They describe that mobile robots face two main problems in navigation. These are localisation, which refers finding ones location and orientation in relation to the surroundings and path planning, which refers to finding a collision free path between two points.

In visual navigation, self localisation is usually done by matching features from the image from the system's camera to features in a map [Bonin-Font et al., 2008].

The way the path planning is execute relies on what kind of navigation technique is used and what the objective of the navigation is.

Tan and Cai [2015] propose dividing path planning algorithms into two categories. The first category is based on environment information directly, while the second uses a structuralised model of the environment. They argue that algorithms that are based on environment information directly most likely will not find the ideal path in a complex environment.

Cherni et al. [2015] describe navigation as either being global or local.

In global navigation the environment is known to the robot and the task is planning the path that leads to the goal while avoiding obstacles in the environment.

Local navigation refers to navigation where the environment is unknown to the robot and it needs to detect obstacles using sensors and to avoid the collision.

Bonin-Font et al. [2008] describe these distinctions as being two ends of a scale ranging from a system acting deliberative or considered. Their distinction is though finer and they establish different groups of navigation techniques.

2.1 Categories of Navigation Techniques

Bonin-Font et al. [2008] categorise visual navigation techniques into two major groups, map-based and mapless navigation. Map-based navigation is further divided into metric map-based navigation, topological map-based navigation, local map-building navigation, and visual sonar techniques. The mapless navigation on the other hand is divided into optical flow-based navigation, appearance-based navigation, image qualitative characteristics extraction for visual navigation and navigation techniques based on feature tracking. This is illustrated in Figure 4.

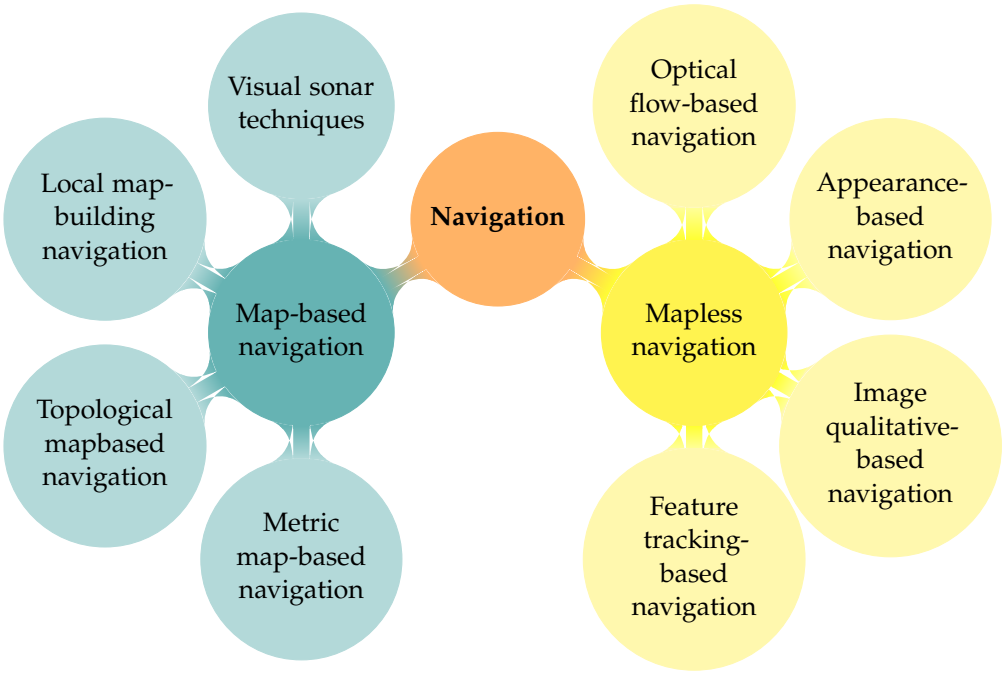

Figure 4: *The categorization of navigation techniques inspired by Bonin-Font et al. [2008]*

In this categorisation map-based navigation relies on building or/and using some kind of map. This means that the map needs to be provided to the system or obtained. The map is either a metric or a topological map. A topological map is a graph-based representation of the environment. In the graph-based representation every node represents a zone in the environment, and can be associated with an action. A metric map includes information such as distances or map cell sizes, while a topological map does not.

Local map-building navigation is distinguished from the two first map-based navigation in the way that it does not use a global representation of the environment. The local map is build instantaneous from the current image of the system's camera. The map has the form of a local occupancy grid, which is used to tell where obstacles are in the field of view of the system's camera.

The last category of map-based navigation techniques are visual sonar techniques, which are used for obstacle avoidance as well. Here the system tries to estimate the distance to the closest potential obstacle and turns away from them. The sonar estimation often works under the premises that the robot is driving on an unicoloured surface.

In mapless navigation no representation of the environment is needed. The system directly takes decisions based on what it perceives.

The first kind of mapless navigation techniques that is described are Optical Flow-Based Navigation Systems. Optical flow is the apparent motion of features in a sequence of images due to the movement of the system itself. Hereby the direction and magnitude of the translational is calculated for the features. The overall logic that is used for Optical Flow-Based Navigation Systems is that the system should turn away from zones of high optical flow. The assumption that this is based on is that objects closer to the system produce optical flow with at greater magnitude than objects far away. This is inspired by the navigation that honeybees use.

Appearance-Based Navigation, which also is a mapless navigation navigation technique, works in two phases. The first phase is called Pre-training, in which images or prominent features of the environment are recorded along with localisation information and/or with an associated control steering command. In the second stage, the correct searing control is found by matching the current obtained image of the system to the images of that were recorded during the pre-training phase. Appearance-Based Navigation can be further divided into two approaches. The first relies on image matching while the second relies on feature matching in order to localise itself and to choose the appropriate steering command.

Image Qualitative Characteristics Extraction is a mapless navigation technique in which pixels are either classified as obstacles or free space. For this two approaches exist. One is the model-based approach in which one or more pre-defined models of known objects exist. The other is the sensor-

based approach in which the sensor information is used directly to do the classification.

The last mapless navigation category are navigation techniques based on feature tracking. Unlike the other techniques, feature tracking-based navigation does not incorporate obstacle avoidance. Instead the obstacle avoidance needs to be implemented with a different method. The robot could for instance track ground space and navigate towards the free space.

2.2 Model for Navigation

From the categorisation of navigation techniques from [Bonin-Font et al., 2008] it can be seen that navigation techniques can have very different objectives and that they make different assumptions about what the system knows and what it does not know. While some researches are mainly concerned about how to obtain a representation of the environment Zhao and Chen [2016] other just assume that a representation is provided to the system Tan and Cai [2015] or have a way of navigating without it.

In order to make this concept more clear the model shown in Figure 5 was developed in this work. It shows all the tasks that a research in the area of mobile visual navigation could be concerned about. The tasks are shown with small boxes. The big surrounding boxes group these into the main choices that are taken in a research. There are two main objectives which are goal finding and obstacles avoidance. Most researcher are concerned about one of these or both. The other groups are local navigation techniques and global navigation techniques. The arrows between the tasks show the order between them. The idea with this diagram is to classify other navigation techniques in terms of which tasks they solve.

There are basically three ways to handle a task for navigation. The task is accomplished by the system, it is assumed to be accomplished by another system or the navigation task is constructed in such a way that it is obsolete. Examples for this are the following related researches.

2.3 Path Planning

The focus of the work of Cherni et al. [2015] is path planning. They assume the system can detect the angle to the borders of the obstacles and that the angle to the goal position can be obtained. They also assume that the system knows how to steer left and right.

In Figure 5 this research is both concerned with goal finding and obstacle avoidance. This is addressed with local navigation. Which means that their developed algorithm can be used for autonomous mobile robot navigation which works in a dynamic and unknown environment. Since the detection tasks both are solved and the research is not concerned about how the locomotion of their robot works it is only concerned about the path planning.

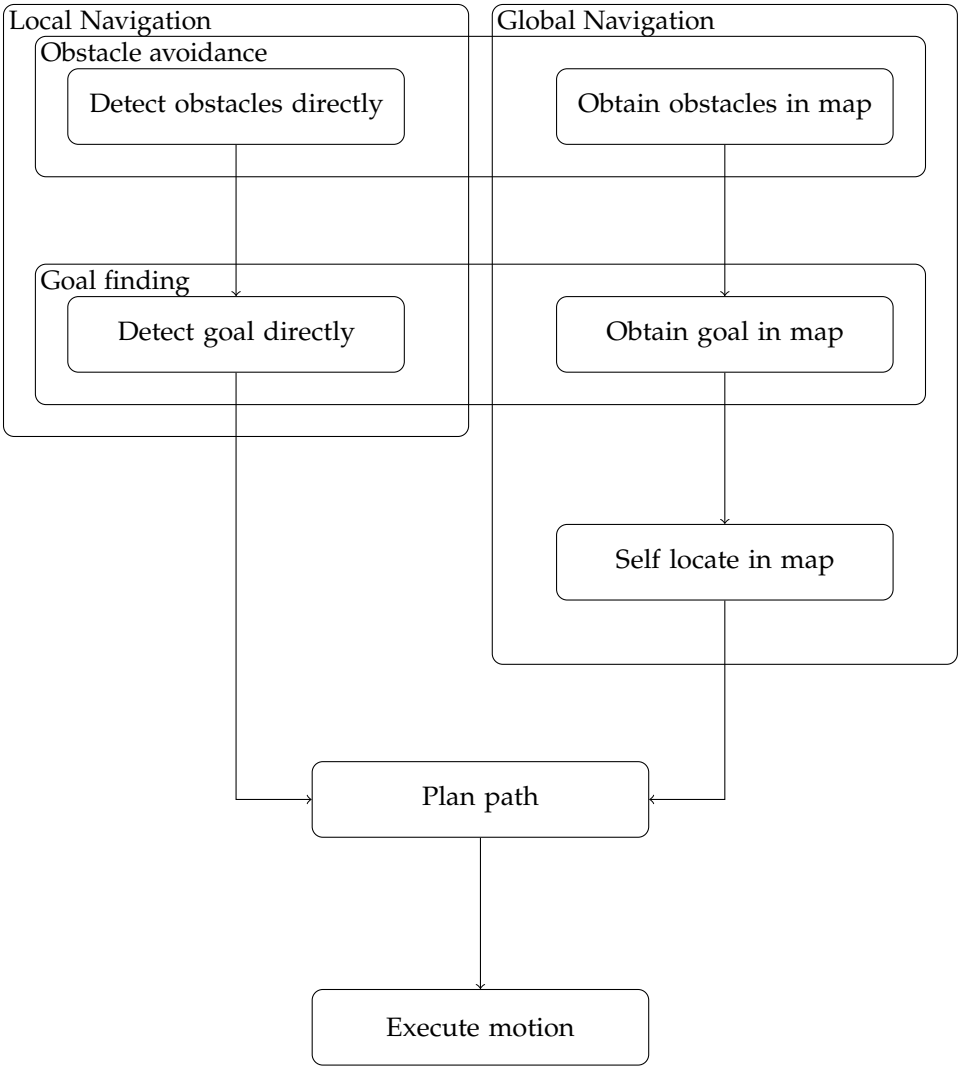

Figure 5: *The tasks that need to be solved in a typical navigation scenario.*

They describe that the main difficulty in navigation is to find the path from the starting point to the target and at the same time avoid collisions.

They show in their experiments, that their algorithm is capable of solving the problem of path planning. This means that it finds a collision-free path between the starting point and the goal in cluttered environment containing obstacles. This is achieved, by constantly calculating the needed angular velocity in order to steer past the obstacles while moving closer to the goal. This is shown in Figure 6.

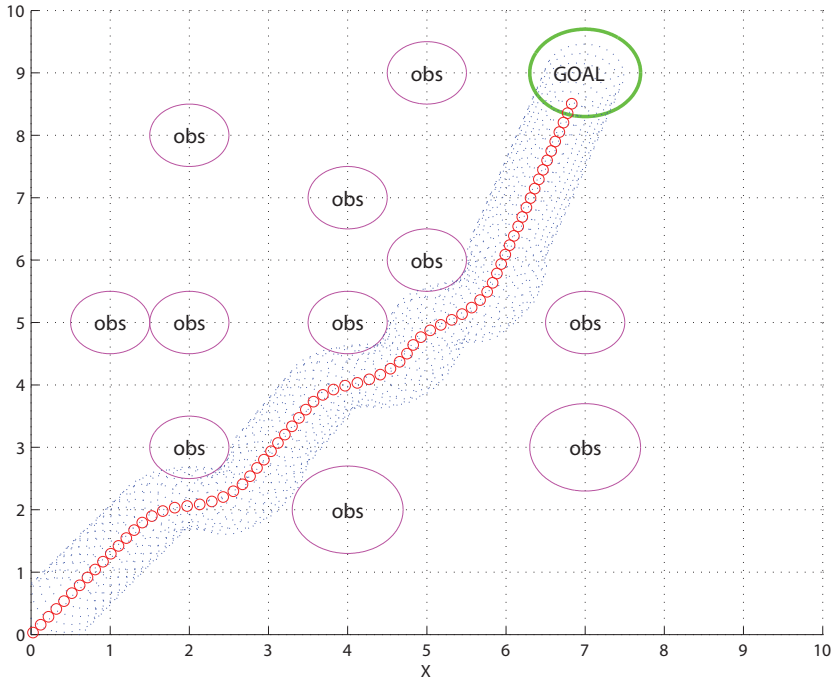

Figure 6: *The path that the robot takes in Cherni et al. [2015].*

Tan and Cai [2015] also focus on path planning. Their algorithm needs to be provided with a global map that contains the obstacles and the goal position. Furthermore some kind of self localisation is assumed to be employed in order to locate the robot on the map. It is also assumed that the system knows which motor controls to use in order to steer the robot to a new position. This means that their algorithm solves the same task in Figure 5 as Cherni et al. [2015], just that they are using Global navigation.

The system is built with an improved Approximate Voronoi Boundary Network (AVBN) and an improved D* algorithm. The map that is used in the system is a 400x400 grids environment, where cells are either occupied or free. The AVBN is then created by eroding the free areas until there is no free area

left. The places where the occupied areas intersect is the path on which the system should be able to navigate safely. These are the red lines in Figure 7. The navigation works by finding the shortest path from the starting position to the intersection area and the shortest path from the intersection area to the goal position. Every place where more than two occupied areas intersect is labeled as a node. These are the blue dots in Figure 7. Additionally the point where the robot has to enter the path and the point where it has to exit the path are labeled as nodes.

The system then has to search for the shortest path from the node where it has to enter the path to the node where it has to exit the path. This is the path from 0 to 9 in Figure 7. This search is then solved with an improved version of the D* algorithm which is called reverse D*.

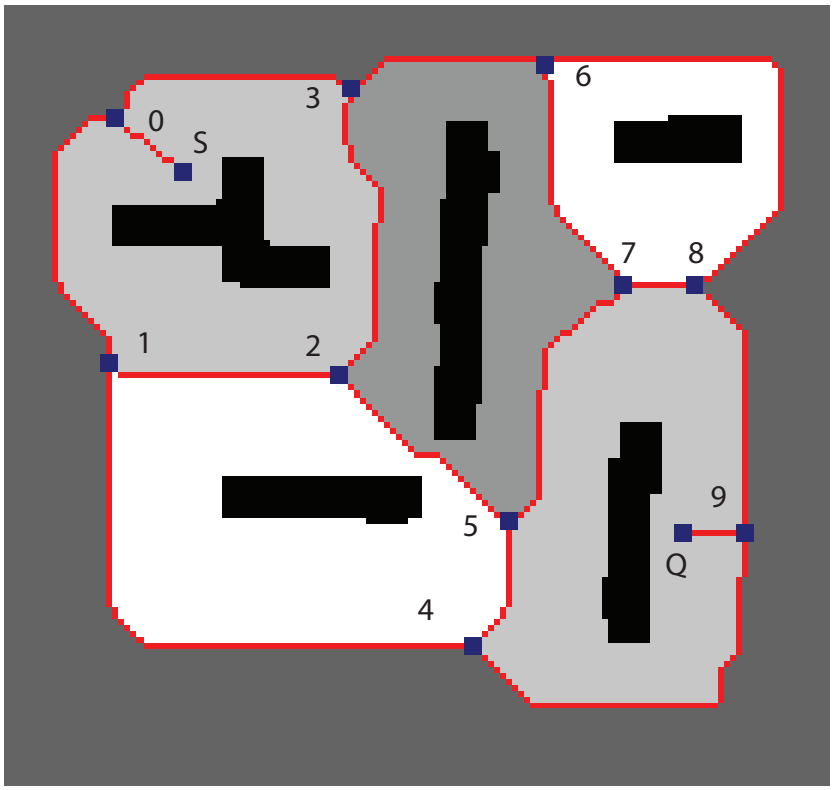

Figure 7: *The AVBN in Tan and Cai [2015].*

2.4 Learning

Yusof et al. [2015] point out that transforming human expert knowledge into computer program only allows a system to solve foreseen and tested outcomes

compared to a system having self-learning capabilities. Therefore unanticipated situations will lead autonomous systems to fail when lacking self-learning capabilities. Therefore they describe that a self-learning system is a more viable solution compared to a system relying on translating pre-defined expert knowledge into computer program.

The pre-defined expert knowledge could be in the form of a map if the ”Obtain obstacles in map” and ”Obtain goal in map” in Figure 5 are solved beforehand. Another form of pre-defined expert knowledge could be a model over the locomotion of the system, which would mean the ”Execute motion” task in Figure 5 is solved beforehand.

A Self-Learning System (SLeS) extract knowledge and learns from past experience. It can thereby discover, classify and memorise new knowledge, which is used to draw inferences.

This kind of behavior is often implemented with reinforcement learning. One of the main differences to other machine learning algorithms is that reinforcement learning does not require any prior training and instead learns from experiences while trying to solve a problem. One of the examples that often is used to show and compare different reinforcement learning techniques is the light finding task [Velez, 2009], [Dini and Serrano, 2012]. This task itself is a simple navigation task in which a robot autonomously tries to reach a light emitting goal.

Velez [2009] develops an artificial neural network (ANN) that is evolved with NEAT and used to control the locomotion of a robot. It teaches an one armed robot to move both forwards and backwards in order to follow a light (See Figure 8). For this the robot has to learn a pull and a push motion, which it has to use dependent on if the light is behind or in front of it. The used fitness is the displacement of the robot when moving toward the light. The light is stationary until the robot reaches the light, after which the light disappears and reappears on the other side of the robot. At this point the robot has to execute the opposite movement. The input to the network are the angular positions of the servos in the robot’s arm, a force sensor at the end of the arm, a contact sensors on the bottom of the robot’s chassis, wheels mounted at the back of the chassis (odometer), and a light sensor. The light sensor is a binary sensor, which only shows if the light is in front or behind the robot. The evolution of the neural network is deemed unrealistic to run on a real robot and is therefore carried out in a simulator. It was found that some of the neural networks that worked well in the simulator did not work well when ported to the real robot.

In terms of Figure 5 Velez [2009] use Local Navigation and are only concerned about Goal finding. The goal detection is not relevant since the system runs in a simulator. The path planning is solved implicitly by directly mapping sensor reading to motion controls. The core of the research is therefore motion execution.

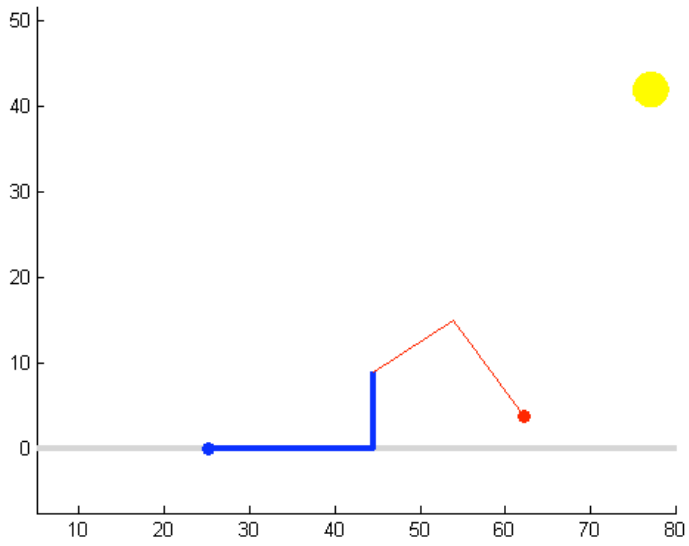

Figure 8: *The simulated robot used in Velez [2009].*

Dini and Serrano [2012] approach the light seeking task with Q-learning (See Figure 9). They propose approximating the action-value function with a feed forward artificial neural network. This network's input neurons take the variables that describe the state that the robot is in as well as the variables that describe the actions that the robot could take. The output of the neural network is a single neuron that gives the approximate output of the action-value, which is the expected utility of performing the given action in the given state. They claim that approximating the action-value would especially be an advantage in larger problems. This addresses the size of the normally required Q-table which holds expected utility and grows exponentially with the complexity of the problem. Both leads to an exponential growing memory demand and an exponential growing convergence time as either the state or the action space grows. Experiments are both conducted in a complex and a simple setting and with standard Q-learning and approximate Q-learning. All experiments are conducted in a robot simulator called pyrobot.

The system is very similar to the system from Velez [2009] in terms of Figure 5. The only thing that is added are sensors to detect obstacles which means that the research is also concerned about the task of ”Obstacle avoidance”.

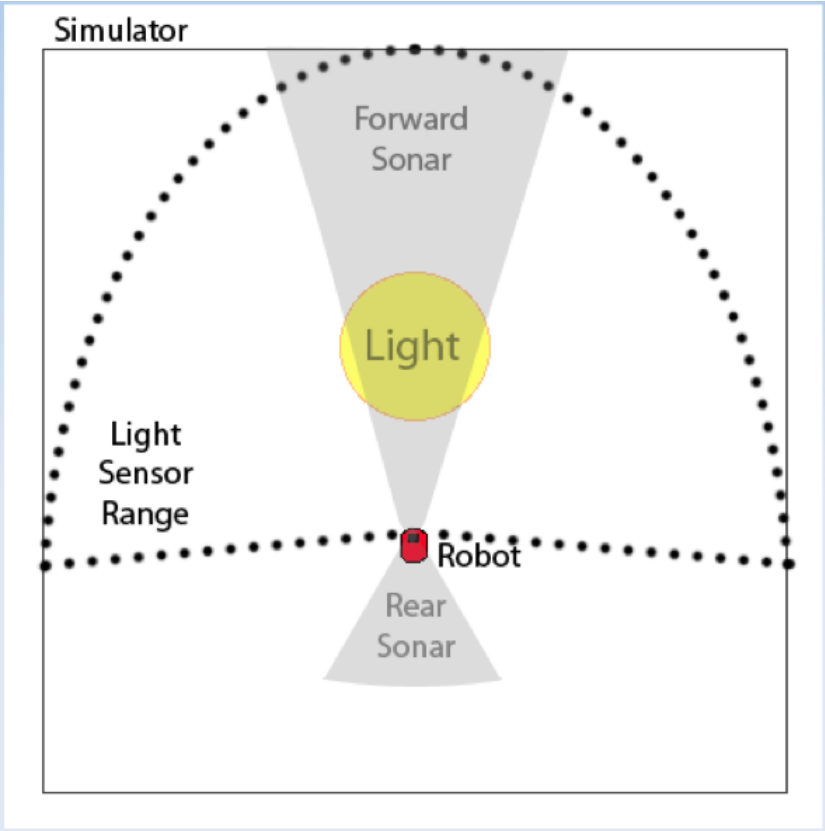

Figure 9: *The simulated light seeking robot in Dini and Serrano [2012].*

Yusof et al. [2015] develop a system that learns to avoid obstacles in an unknown simulated environment. In the environment the system controls a two-wheeled robot with a thirteen sonar sensors which provide an approximation of the distance into different directions. The robot can be seen in Figure 10.

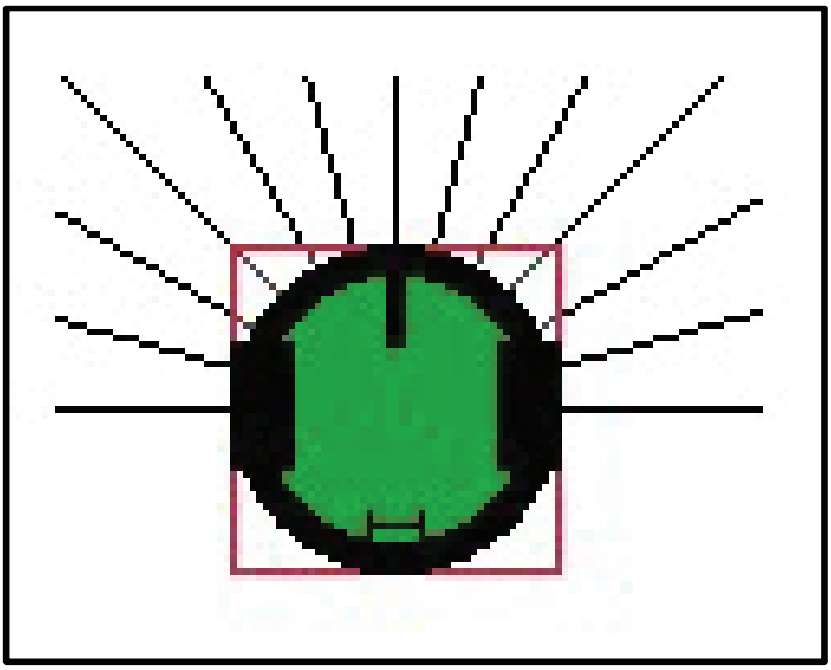

Figure 10: *The simulated robot with the thirteen sonar sensors from Yusof et al. [2015].*

The readings from the sensors are used as input to an unsupervised weightless neural network learning algorithm called AutoWiSARD. This neural network is used to identify, differentiate and classify the obstacles in the environment. The idea is that it is supposed to handle the problem of generalisation. In order to react to the detected obstacles with an action the reinforcement learning technique Q-learning is used. Q-learning reacts to a so called state which in this case is the obstacle detection from AutoWiSARD with a specific action. It is then rewarded or punished by a reward mechanism. In the work of Yusof et al. [2015] the system was rewarded with a value of 1 for driving forward and not hitting an obstacle, punished with a value of -0.7 when an obstacle was hit and slight penalties of -0.1 and -0.3 were given for turning into the same direction and the opposite direction respectively.

The system is therefore only concerned about obstacle avoidance and not goal finding when described with Figure 5. The path planning is also included in the motion execution, since detected obstacles are mapped directly to movement primitives.

Drchal et al. [2009] use a recurrent neural which is evolved with HyperNEAT to control mobile driving robots in a robot simulation (See Figure 11). All robots are equipped with the same neural network. The task of the robots is to drive and stay on the road. The chosen fitness is the maximum average speed

of the robots. The robots' speed is reduced when not driving on the road. The maximum speed therefore leads to the development of a network that lets the robots stay on the road while driving fast. The input to the robots' networks are sensors organised in polar coordinates in two quadrants in front of the robot. These are binary sensor which represents the surface friction (road or no road). The network has two output neurons, which activations are used to set the wheels acceleration. One of the future plan for the project is to use real robots with omni-directional cameras as input devices.

It is claimed that the extension to omni-directional camera should be straightforward, even if the surface detection is not explained in their work.

Since the robots described by Drchal et al. [2009] are only concerned about driving fast and not towards a goal the system's objective is obstacle avoidance. Their system is a special case since the obstacles are in the form of surface with a higher friction. Like the other described learning system is the path planning done implicitly by executing motion directly.

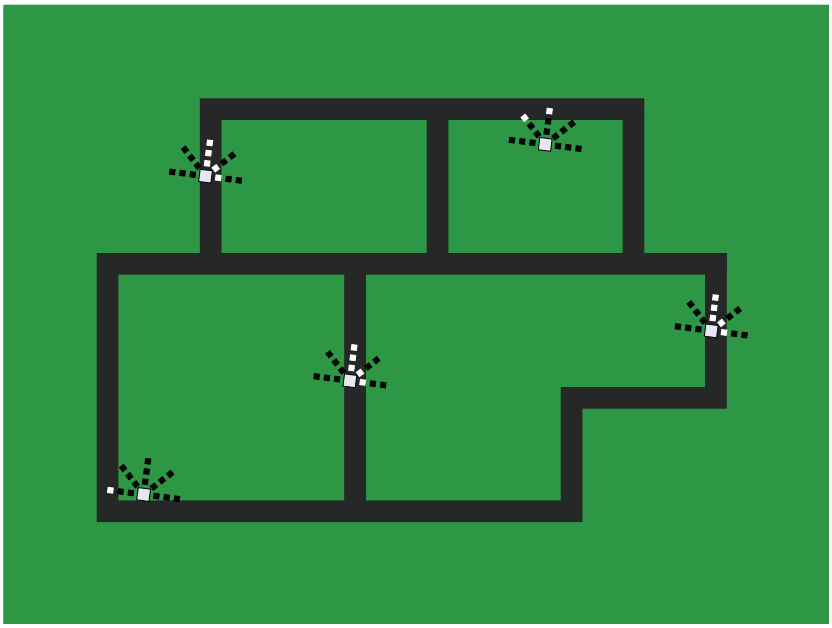

Figure 11: *The simulated mobile robots used by Drchal et al. [2009].*

Gaskett et al. [1999] describe a method to use Q-learning for control tasks which require continuous actions, in response to continuous states. This is done by using only the state of a system as the input to a neural network. The output of the neural network are samples from the action-value space, which are used to interpolate the entire space. The developed system is used to

control a simulated submersible vehicle to a target position by firing thrusters located on either side of the vehicle (See Figure 12). The system is given 200 time steps to reach the target position after which a new target is set. It is shown that the system successive get better at reaching the targets. The performance is even increased by Advantage Learning, which is a variation of Q-learning.

There are no obstacles in the test setup from Gaskett et al. [1999] the objective of the research is therefore goal finding. Since no sensors are described and the system is only running in a simulation with a known relative goal positions it could either be used in global or local navigation.

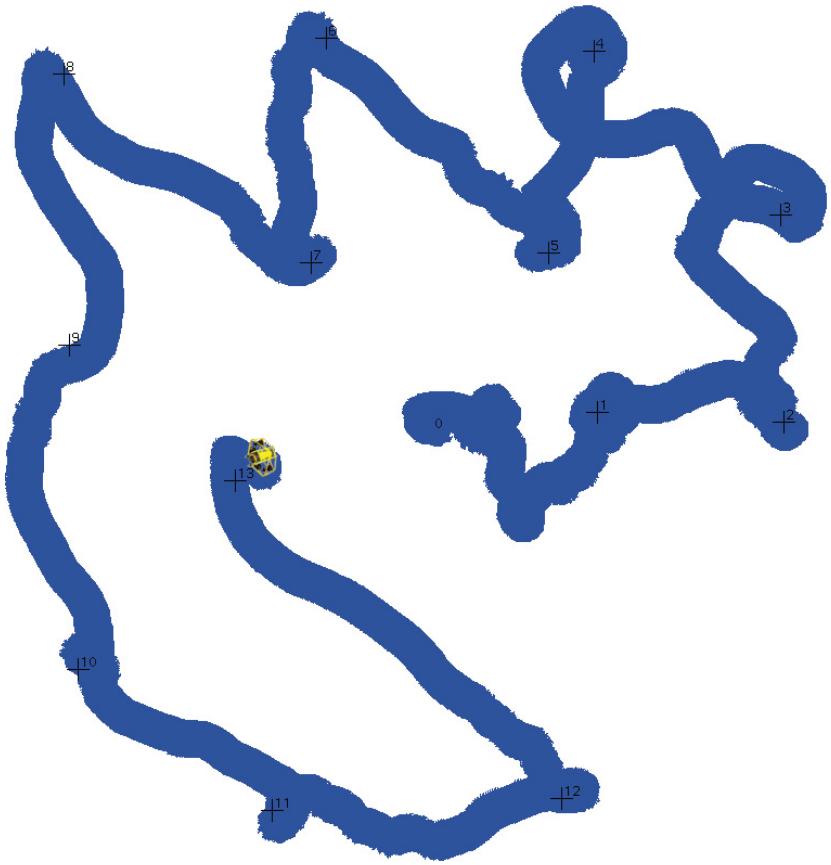

Figure 12: *The route the simulated submersible vehicle has taken in the test in Gaskett et al. [1999].*

Bischoff et al. [2013] suggest to use hierarchical reinforcement learning to solve a navigation task (See Figure 13). In hierarchical reinforcement learning

the original task is split into elementary subtasks in order to limit the search space. The first subtask is to take the robot's grid-cell position as the state of the system and predefined movement primitives as action space. The goal of the first subtask is to learn a policy that brings the robot to the goal cell. The second subtask is to learn the execution of the movement primitives. In this work the translation of the robots, x, y-position, and yaw Î¸ is the state space and the the rotational speed of each three wheels is the action space. While the first subtask can be solved in a discrete state and action space the second is solved with continuous reinforcement learning. It is shown that the chosen hierarchical architecture is suitable for learning robot navigation. The learned policy can be updated online and therefore takes dynamic obstacles into account. The position of the robot is determined by a tracking system rather then a camera on the robot as in the other projects.

Bischoff et al. [2013] both incorporate obstacle avoidance as well as goal finding in their first subtask. The second subtask solves the movement execution in Figure 5.

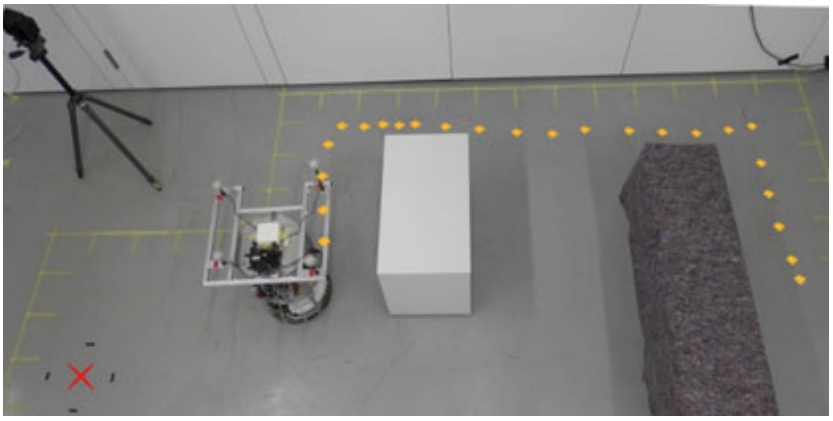

Figure 13: *The testsetup used in Bischoff et al. [2013].*

3 Analysis

As it is established in the initial problem statement, is a system in this project sought that is not specific to a robot platform. By looking at the work that is related to navigation, it is clear that a lot of very specific solutions exist, which accomplish very specific tasks in navigation.

In order to develop a system, that can be used in a variety of navigation situations, as little assumptions as possible are made about what knowledge is available to the system. Assuming that the system gets environment representation provided is therefore not a viable solution. It is chosen to

develop a system that can operate in an unknown environment. The same counts for the locomotion of the system. It is therefore not assumed that the system knows how motor commands relate to self movement.

The seeked system needs to use a local navigation method and it needs to learn how the perceived relates to the motion that needs to be executed. By learning this relationship on the fly, the system won't need any prior knowledge and unanticipated situations can also be handled by the system. Looking at the related work, reinforcement learning seems to be a good solution for this kind of problem.

3.1 Simulations

The described related projects that use NEAT are both used in a simulator. As opposed to most neural networks which are trained with the backpropagation algorithm, NEAT is trained with an evolutionary algorithm. For this kind of training the performance of different neural network setups is evaluated over a period of time. The best networks are then combined into new networks and the evolution is started over again. The advantage of this approach is that the weights as well as the topology (the layers and number of neurons in each layer) is evolved. The disadvantage of this approach is that it requires running multiple systems for a period before any adjustment is made. Stanley and Miikkulainen [2002], who are the inventors of NEAT use the system to learn an XOR gate for verification. For this task, the system starts with two input neurons, one bias neuron and one output neuron and should learn to use a hidden neuron. In his experiment the system is run 100 times, in which it finds a structure for XOR in on average 32 generations, with 4755 networks networks evaluated, (std 2553). A navigation task is most likely more complex than an XOR gate and would therefore on average require more than 4755 evaluations over a period of time and therefore probably was deemed unrealistic by Velez [2009].

Furthermore it is not possible to reset the real world to give all network the same starting condition as it is in the simulations, which would probably increase the training time. Drchal et al. [2009] claim the extension of their project to a real environment, should be straightforward with a omni-directional camera. Velez [2009] tried to transfer to their trained system to a real robot, which didn't work. This might be interpreted the real world being more complex than a simulator. This is why reinforcement learning techniques that are meant to be used in the physical world should be tested in the physical world in order to remove the ambiguity to the simulated and ensure actual the applicability on real task. Furthermore the idea behind reinforcement learning is that system learns from experience and is not pre-trained in a simulator.

3.2 Continuous Spaces

One of the most common reinforcement learning techniques is Q-learning [Stone, 2014]. Q-learning works by estimating the value for a given number of discrete action executed from a number of discrete states allowing a system to choose the action that yields the highest value. Reinforcement learning with NEAT is able to map continuous states to continuous action, which is useful in a lot of different navigation tasks. In order to use Q-learning with continuous state, Gaskett et al. [1999] and Bischoff et al. [2013] use Neural Networks. This has the advantage that their system can generalise over similar states. It is important for Q-learning that the maximum value in a state can be found. Bischoff et al. [2013] achieve this by feeding the state together with every possible action through the network. However, its disadvantage is that the actions have to be finite, which is not the case for a continuous action space. Gaskett et al. [1999] feed the state into the network, but use a novice interpolation technique which can predict any value for any action in the given state from a few sample the action values space. The used interpolator is designed in a way that the highest interpolated value always coincides with highest sample value. This allows Q-learning with continuous actions.

3.3 Generalisation

While generalisation in some parts of navigation is essential for fast learning, it is in other parts fatal. Bischoff et al. [2013] solve this, by splitting the learning into how to execute movement primitives (generalised) and where to execute which movement primitives to avoid obstacles (no generalisation). Compared to the other described projects, the robot's position is determined with a tracking system. While this would work for specific applications of autonomous navigation is there a wide variety where this would not be possible (e.g. flying a drone outdoor, driving autonomous car etc.). This input is essential as a feedback for learning the movement primitives.

In the test performed by Gaskett et al. [1999], the submissive vehicle learns how to set its thrusters to get to a specific goal position. This could be seen as movement primitive, which could be combined with hierarchy learning presented by Bischoff et al. [2013].

3.4 Final Problem Statement

The goal of this project is to find a reinforcement learning algorithm, that can be used for autonomous navigation based on the input from a camera. This should be implemented as a self-learning system in order to be applicable in a variety of navigation problems and work on different robot platforms. This has the advantage that the system does not have to be pre-trained i.e. learns from experience and make less mistakes over time.

From the analysis of the related work can be seen that directly using NEAT for reinforcement learning does not yield good results in a real environment.

Furthermore it is established that directly using Q-learning brings some undesirable limitations to the system.

It was therefore chosen to focus on approximate Q-learning. Since Gaskett et al. successfully implement approximate Q-learning for a continuous action and state space, their approach is used as a starting point for this project.

Gaskett et al. [1999] show that their system is capable of learning to navigate using approximate Q-learning. Even though this is established to be successful would it be of interest to now how good the Q-function actually is approximated, which is why one of the research question is:

- How well is the Q-function approximated by Gaskett et al.'s method?

Since Gaskett et al. [1999] only test their system in a simulated environment one of the research questions of the here presented work is:

- Is the adapted system working in a non-simulated environment?

Furthermore it is of interest if the system would work in a scenario where it perceives the world via a camera, which is why the next research question is:

- How can Gaskett et al.'s method be adapted in order to work with computer vision on an embedded platform?

4 Theory

In this section the theories and concepts that are used in this project are explained. All symbols in the following equations are adjusted to match the work described by Watkins and Dayan [1992] in order to keep the notation consistent.

4.1 Reinforcement Learning

In Reinforcement Learning a training dataset is generated by an agent's interactions with the environment. This is different from most machine learning algorithms in the way that the training-set normally is given beforehand. In response to an action taken by the agent, the environment generates an observation and an instantaneous cost or reward according to some (usually unknown) dynamic. This is illustrated in Figure 14.

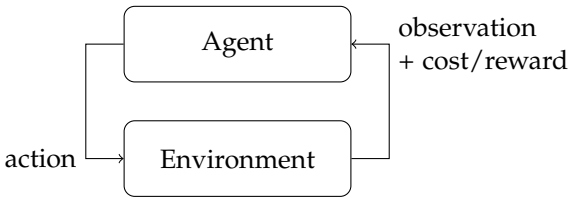

Figure 14: *Reinforcement learning.*

The agents task is to select actions that minimises some measure of a long-term cost [D'Addona, 2014].

The environment in Figure 14 is often formulated as a Markov decision process.

A Markov decision process can mathematically be described by:

- a set of states X.
- a set of a actions that can be taken from the states A.
- the probabilities $P_{x_n,y}[a_n]$ of landing in state y after having taken action a from state x at any time step n.
- the rewards $R_x(a)$ for taking action a from state x.
- the discount factor $\gamma \in [0, 1]$ which describes how much the rewards importance is decaying over time.

[Watkins and Dayan, 1992]

In the Markov decision process, the probabilities $P_{x_n,y}[a_n]$ of landing in a state y are only conditioned on the previous state x and the taken action a. They can therefore be rewritten as shown in eq. (1). This is called the Markov property [Emigh et al., 2015].

$$P_{x_n,y}[a_n] = P(y_n = y \mid x_n, a_n) \tag{1}$$

The long term cost in the Markov decision process is minimised by maximising the discounted sum of future reward given by eq. (2). This function is not explicitly mentioned by Watkins and Dayan [1992], but can be derived from their description and gains the understanding of the following.

$$\sum_{n=0}^{\infty} \gamma^n R_{x_n}(a_n) \tag{2}$$

The agent's task is to determine an optimal policy $\pi^*(x)$, which maximises the sum in eq. (2). The asterisk specifies that the policy is optimal and not just any policy $\pi(x)$

The policy describes which action a should be executed in which state x.

It is important to note that eq. (2) contains all rewards that the system will get in the future and not only the immediate reward. This is essential when the optimal policy requires the agent to take actions for which it receives a low immediate reward, but enters states from which high rewards can be received.

Since rewards that are received sooner than later are more valuable, the future rewards are discounted with respect to how far they are expected to be received in the future. This is controlled by the discount factor γ, which is used to weight each reward by its delay. γ is usually chosen to be between 0 and 1. A γ of 1 would mean that future rewards are as important as the immediate rewards and 0 would mean that they have no impact on the agents decision.

An Markov decision process is considered to be solved when the optimal policy $\pi^*(x)$ is found, which means that it maximises eq. (2) from any state. [Emigh et al., 2015].

The two main approaches for this are model based and model free reinforcement learning. In model based reinforcement learning, the probabilities $P_{x_n,y}[a_n]$ and the rewards $R_x(a)$ from the Markov decision process are learned. When these are considered to be known, the policy can be formulated offline using methods such as value iteration and policy iteration [Emigh et al., 2015].

These rely on evaluating the policy and the value for all states with eq. (3) and eq. (4). In this process an array of actions and an array of values is kept, which is updated with eq. (3) and eq. (4) until the policy values stop changing.

$$\pi(x) = \arg\max_a \left\{ \sum_y P_{x_n,y}[\pi(x)] \left(R_x(\pi(x)) + \gamma V^\pi(y) \right) \right\} \tag{3}$$

$$V^\pi(x) = \sum_y P_{x_n,y}[\pi(x)] \left(R_x(\pi(x)) + \gamma V^\pi(y) \right) \tag{4}$$

In policy iteration [Howard, 1960] eq. (3) is evaluated once for all states after which eq. (4) is evaluated until the values stop chaining. These two steps are repeated until also the policy stops changing.

In value iteration [Bellman, 1957] eq. (3) and eq. (4) are updated together, by repeatedly evaluating eq. (5).

$$V^\pi(x) = \max_a \left\{ \sum_y P_{x_n,y}[\pi(x)] \left(R_x(\pi(x)) + \gamma V^\pi(y) \right) \right\} \tag{5}$$

In model free reinforcement learning the policy is learned directly without learning the probabilities $P_{x_n,y}[a_n]$ and the rewards $R_x(a)$.

4.2 Q-learning

Q-learning is one of the most commonly used reinforcement learning techniques [Stone, 2014]. It is model free since $P_{x_n,y}[a_n]$ and $R_x(a)$ are not computed.

Instead the action-value function is learned, which is also called Q-function. This function returns a value for every action from every state of the Markov decision process. The function, from Watkins and Dayan [1992] can be seen in eq. (6).

$$Q^{\pi}(x,a) = R_x(a) + \gamma \sum_{y} P_{xy}[\pi(x)]V^{\pi}(y) \tag{6}$$

When the Q-function is known for every action from every state, the optimal policy corresponds to executing the action in every state that yield the highest value.

In standard Q-learning a look-up table representation of the Q-function is used. This table has one field for every possible combination of an action that can be taken from a state. This is illustrated in Figure 15. This table is usually initialised with small random numbers. The agent therefore chooses random actions when presented with a state for the first time.

	x	y	z
a	$Q(x,a)$	$Q(y,a)$	$Q(z,a)$
b	$Q(x,b)$	$Q(y,b)$	$Q(z,b)$
c	$Q(x,c)$	$Q(y,c)$	$Q(z,c)$
d	$Q(x,d)$	$Q(y,d)$	$Q(z,d)$

Figure 15: *Look-up table representation of the Q-function.*

The procedure that an agent in Q-learning has to execute consists of determining in which state the system is, choosing an action, determining in which state the system landed and learning from the received reward. The update of the Q-function is described by Watkins and Dayan [1992] and shown in eq. (7). The time of each variable is given by their subscript, n denotes any point in time of the learning process and α the learning rate.

$$Q_n(x,a) = \begin{cases} (1-\alpha_n)Q_{n-1}(x,a) + \\ \quad \alpha_n[r_n + \gamma V^{\pi}_{n-1}(y_n)] & \text{if } x = x_n \text{ and } a = a_n \\ \\ Q_{n-1}(x,a) & \text{otherwise} \end{cases} \tag{7}$$

where

$$V^{\pi}_{n-1}(y) = \max_{b}\{Q_{n-1}(y,b)\} \tag{8}$$

The learning process in eq. (7) is mimics value iteration.

In the procedure a prediction is update with a target value. The prediction value is the until that point believed true value, in this case $Q_{n-1}(x,a)$, and

the target value is the new believed true value, in this case $r_n + \gamma V_{n-1}(y_n)$. The Q-value is updated by a combination of the new and the old value. The learning rate α is used to regulate how much of the old and how much of the new value is used.

A desirable property of Q-learning is that it works without any prior training of the system, since the relationship between states actions and rewards is learned on the fly. Additionally the system will be able to adapt to a changing environment and adjust the model of its surrounding if the current model is contradicted.

4.3 Exploration

While it might be that it seems like a good idea only to choose the action that yields the highest Q-value, is likely that the agent will fall into a local maximum of the total discounted expected reward, while the Q-function is being learned.

The question here is if the agent should take the action that looks the most appealing according to the model or if it should try out something new.

Choosing only the actions that yield the highest Q-value of the current model is called exploitation while choosing an actions that according to the current model is suboptimal is called exploration. The problem with only exploiting is that the agent will not learn about actions that are assumed to be suboptimal, when they are really unexplored.

This could be due to the fact that the environment has changed or that the agent has not tried the action often enough to have a good representation of the Q-value.

In order to balance exploration and exploitation an exploration function is used. The exploration function decides which action to take based on the Q-value of the available actions and other parameters.

ε-greedy is one of the simplest exploration functions, which chooses the optimal action and with a probability of ε a random action.

A more sophisticated method is the soft-max action selection. In this method the probability of taking an action is the fitness of that action divided by the sum of all action's fitnesses. The fitness in this case would be the Q-value. The probability of taking an action a is given by eq. (9), where T is used to control the amount of exploration [Sutton and Barto, 2005].

$$p(a) = \frac{e^{Q(a)/T}}{\sum_{b=1}^{n} e^{Q(b)/T}} \tag{9}$$

4.4 Approximate Q-Learning

One problem that exists with standard Q-learning is that it does not generalise across similar states or actions. That means that if the action space is extended

by only one action, the action state space grows by the number of states and vice versa.

The convergence time therefore grows exponentially with the size of the reinforcement learning problem that is to be solved.

Q-learning is therefore limited to problems of a certain size. To avoid this problem, one can approximate the Q-function. Rather than learning a lookup table for every possible scenario (consisting of a state and an action) one can learn how to weight features that describe the scenario in order to get the Q-value. The weights of the features can be updated in the same way as the Q-value for a single scenario.

The simplest form of learning how to weight the features is called linear regression. In order to learn more complex non-linear relations a neural network can be used.

It is worth noting that Watkins and Dayan [1992] assume a look-up table representation of the Q-function in order for it to converge. Furthermore, its convergence is only guaranteed if the learning includes an infinite number of training samples, while the episodes need not to be consecutive.

4.5 Neural Networks

In the problem domain of supervised learning, artificial neural network (referred to as neural network in the context of machine learning) are used to learn a function that maps $X \rightarrow Y$ given pairs of (x, y), where $x \in X$ and $y \in Y$. The main advantage of neural networks is that they can approximate non-linear relations. This is especially useful when the complexity of data or task makes the by hand design impractical. Neural networks consist of interconnecting artificial neurons which are programming constructs that mimic the properties of biological neurons[D'Addona, 2014]. An example of this can be seen in Figure 16, where each grey circle symbolises an artificial neurons and each arrow an interconnection.

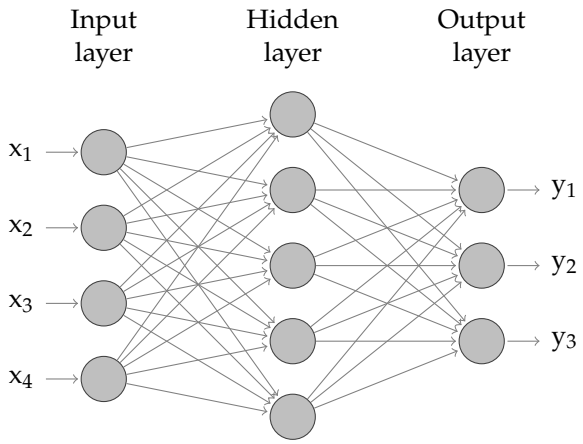

Figure 16: *An example of a neural network structure.*

A widely used type of composition for neural networks is the non-linear weighted sum [D'Addona, 2014]. For Figure 16 this is shown in eq. (10), where $y_j(x)$ would be the output of any neuron j, x would be the input vector to the network, f_j the neuron's activation function, I the subset of neuron connected to j and W_{ji} is the weight connecting neuron j to i.

$$y_j(x) = f_j(\sum_{i \in I} W_{ij} y_i(x)) \tag{10}$$

From eq. (10) the similarity to biological neurons can be seen. Biological neurons communicate with each other by sending electronic pulses to each other. The input to the function can be seen as frequencies of these pulses. After a biological neuron has received enough input pulses, it will output a pulse to other neurons. This is modelled by a non linear activation function [Vreeken, 2003].

Back propagation

The back propagation algorithm can be used to adapt the weight of a neural network in order to perform a mapping from $X \rightarrow Y$ given a training set of pairs of (x, y) where $x \in X$ and $y \in Y$. Back propagation is a gradient-descent algorithm, which means that it uses the gradient of the error derivatives with respect to the weights to reduce the error the system makes.

For this the error is described as the function of the system's weight vector of the system $E(\mathbf{w})$. By moving the weight vector in to the opposite direction of the gradient vector the steepest error decrease can be achieved Nikolaev and Iba [2006].

The back propagation can be described in two passes a forward pass and a backward pass. In the forward pass the outputs of neurons in the network

are computed. In the subsequent backward pass the error made by the system is back propagated and the weight are updated with in the direction of the greatest error descent.

A commonly used cost function to determine the error is the mean-squared error function [D'Addona, 2014], which is shown in eq. (11). The $\frac{1}{2}$ is added, so the derivative becomes $(y(x) - t)$. Here t is the target value from the training dataset and $y(x)$ is the output of the system when given the x as input.

$$E = \frac{1}{2}(y(x) - t)^2 \tag{11}$$

The partial derivative with respect to any weight from neuron i to j is shown in eq. (12).

$$\frac{\delta E}{\delta W_{ij}} = f_i(x_i) \cdot delta(j) \tag{12}$$

where

$$delta(z) = \begin{cases} f'_z(x_z) \cdot (y_z(x) - t_z) & \text{if } z \in \text{output-} \\ & \text{neurons} \\ f'_z(x_z) \cdot \sum_{m \in M} delta(m) \cdot W_{mz} & \text{otherwise} \end{cases} \tag{13}$$

The error derivatives shown in eq. (12) were derived from the functions in Nikolaev and Iba [2006] to better show the recurrency of the part of the called $delta(z)$ shown in eq. (13). In the functions, x is the input to any neuron, f is the activation function, f' is the derivative of the activation function, and M is the subset of neurons sending to neuron z.

The algorithm can be implemented in batch mode or in incremental mode Nikolaev and Iba [2006] as follows:

In the batch mode the error derivatives with respect to the weights are accumulated for the entire training set. These accumulated error derivatives are used to perform one training epoch with gradient descent. The error that is minimised is the error made on the entire set. This makes the update little sensible to outliers and noisy samples in the training set.

In incremental back propagation, the weights are updated iteratively by finding the error derivatives for one sample at a time and subsequently performing an update. The error that is minimised is the error made on the single sample. This mode converges slower than batch, but is better at escaping poor local minima on the error surface for the entire sample.

Momentum

The momentum method is a method that improves the convergence time, by lowering oscillates of the weight updates. This is done by updating the weights with the current derivatives and the attenuated last update. In eq. (14), the w_n

is the weight at time n, Δw_n it the update that would have been performed without momentum and $\alpha \Delta w_{n-1}$ is the attenuated update from previous time step Nikolaev and Iba [2006].

$$w_{n+1} = w_n + \Delta w_n + \alpha \Delta w_{n-1} \tag{14}$$

This method cancels a potentially oscillating part of Δw_n out, while building up momentum in the non oscillating direction.

4.6 Continuous States and Actions

A lot of problems in the real world require a system to respond to continuous states with continuous actions. Since continuous features can be used in the function that is used to approximate the Q-Value, this problem gets partially solved.

As it can be seen in eq. (8) it is necessary for update of the Q-function to find the maximum Q-Value in a given state y. This is not a trivial task, when both the state and the action are used as input to a function that approximates the Q-value and is able to model non-linear relations. Gaskett et al. solve this problem by approximating the entire action value space based on a given state.

The used interpolation scheme is called wire-fitting, which is used to interpolate the value for any action in a given state based on samples for specific actions in the action-value space. The samples are called wires and consist of the value q and the action vector $\mathbf{u}$, given a state $\mathbf{x}$. The interpolation function is shown in eq. (15).

$$Q(\mathbf{x}, \mathbf{u}) = \lim_{\epsilon \to 0^+} \frac{wsum(\mathbf{x}, \mathbf{u})}{norm(\mathbf{x}, \mathbf{u})} \tag{15}$$

where

$$wsum(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{n} \frac{q_i(\mathbf{x})}{dist(\mathbf{x}, \mathbf{u})} \tag{16}$$

and

$$norm(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{n} \frac{1}{dist(\mathbf{x}, \mathbf{u})} \tag{17}$$

where

$$dist(\mathbf{x}, \mathbf{u}) = ||\mathbf{u} - \hat{\mathbf{u}}_i||^2 + c(q_{max}(\mathbf{x}) - q_i(\mathbf{x})) + \epsilon \tag{18}$$

In eq. (16), eq. (17), and eq. (18) i is used to describe the index of the wire. Additionally q_{max} is used to describe the value the highest value among the wires, c is a smoothing constant, and ϵ should avoid the division by 0.

The wires can be updated with gradient descent, similar to the neural network, in order to better match a value after having taken an action. For this, the partial derivatives of the error with respect to the components of the wires need to be found. The mean-squared error function for is shown in eq. (19) and its derivative is shown in eq. (20), where w could be any component of the wires, $Q_p(\mathbf{x}, \mathbf{u})$ it the predicted output, and $Q_t(\mathbf{x}, \mathbf{u})$ is the received output after having taken action $\mathbf{u}$ from state $\mathbf{x}$.

$$E = \frac{1}{2}(Q_p(\mathbf{x}, \mathbf{u}) - Q_t(\mathbf{x}, \mathbf{u}))^2 \tag{19}$$

$$\frac{\delta E}{\delta w} = (Q_p(\mathbf{x}, \mathbf{u}) - Q_t(\mathbf{x}, \mathbf{u})) \cdot \frac{\delta Q_p(\mathbf{x}, \mathbf{u})}{\delta w} \tag{20}$$

The partial derivatives of $Q_p(\mathbf{x}, \mathbf{u})$ are given in eq. (21) in terms of q_k and in eq. (22) in terms of $u_{k,j}$.

$$\frac{\partial Q}{\partial q_k} = \lim_{\epsilon \to 0^+} \frac{norm(\mathbf{x}, \mathbf{u}) \cdot (dist(\mathbf{x}, \mathbf{u}) + q_k \cdot c) - wsum(\mathbf{x}, \mathbf{u}) \cdot c}{[norm(\mathbf{x}, \mathbf{u}) \cdot dist(\mathbf{x}, \mathbf{u})]^2} \tag{21}$$

$$\frac{\partial Q}{\partial u_{k,j}} = \lim_{\epsilon \to 0^+} \frac{[wsum(\mathbf{x}, \mathbf{u}) - norm(\mathbf{x}, \mathbf{u}) \cdot q_k] \cdot 2 \cdot (u_{k,j} - u_j)}{[norm(\mathbf{x}, \mathbf{u}) \cdot dist(\mathbf{x}, \mathbf{u})]^2} \tag{22}$$

Gaskett et al. [1999] state that eq. (21) is inexact when $q_k = q_{max}$. It is though not mentioned what to do about this fact. The magnitude of this problem becomes evident when taking the action that yield the highest reward. In that case the conditions in eq. (23), eq. (24) and eq. (25) would be true.

$$dist_{max} \to 0^+ \quad \text{if } \mathbf{u} == \hat{\mathbf{u}}_i \text{ and } q_i(\mathbf{x}) == q_{max} \tag{23}$$

$$wsummax \to \infty^+ \quad \text{if } dist_{max} \to 0^+ \tag{24}$$

$$normmax \to \infty^+ \quad \text{if } dist_{max} \to 0^+ \tag{25}$$

Under these condition, eq. (21) would become eq. (26).

$$\frac{\partial Q}{\partial q_k} = \lim_{\epsilon \to 0^+} \frac{\infty^+ \cdot (0 + q_k \cdot c) - \infty^+ \cdot c}{[\frac{0}{0}]^2} \tag{26}$$

Gaskett et al. [1999] state that ϵ should avoid the division by 0. One might therefore think that setting ϵ to a small number resolves the problem. This is not the case as can be seen by substituting a large number for ∞^+ and its reciprocal for 0 in eq. (26) which would become a eq. (27) and always be a large negative number.

$$\frac{\partial Q}{\partial q_k} = 1 - 1\frac{1}{\infty^+ + q_k * \epsilon} - \infty^+ * c \tag{27}$$

This problem can be resolved by either setting $c = \epsilon$, which would lead to a derivative of approximate 0 or by just ignoring the the derivative in the gradient descent update.

4.7 Exploration

An aspect that is not really covered by Gaskett et al.'s work is how the system should choose between exploration and exploitation.

It was therefore chosen to try ε-greedy and an adapted version of the soft-max action selection. Standard soft-max action selection only works with discrete actions as seen in eq. (9).

In order to use this kind of mechanism in this project, it was chosen to observe the distance that the wires had to each other. When the system is started all wires are roughly in to the same area around 0. The more detailed the model gets the more widespread the wires have to be. It is therefore chosen to take a random action with the chance given by eq. (28). Otherwise the system would act greedy. The $dist()$ function that is used is the is the euclidean distance and T would again control the amount of random behaviour.

$$p(a_{rnd}) = e^{\frac{\sum_i dist(wire_{max}, wire_i)}{T}} \tag{28}$$

4.8 Parameter Tuning

For the reinforcement learning algorithms, investigated in this project, a variety of different parameters can be set. It is therefore important to access the right parameters for the reinforcement learning algorithms.

The parameters were set with the stochastic search method simulated annealing which can be used for both discrete and continuous optimisation problems [Pardalos and Mavridou, 2009]. This was mainly done to test for the correlated effect of changing parameters together.

The idea with the simulated annealing is that parameter combinations are seen as states with energy. The energy is a measurement of how good the state is. The system can move from a state to a neighbour state, by perturbing the current state.

Simulated annealing will always move to states that have a higher energy level than the current state. It will also move to states with a lower energy level with a probability that depends on the energy difference between the two states. The probabilities for this is seen in eq. (29), where q_t is the current state, r_{t+1} is the subsequent state and $E()$ is the energy function [Pardalos and Mavridou, 2009]. k_b is a physical constant known as the Boltzmann constant. T is used to control the the amount the probability to moving to a state with a lower energy and should be decreased over time.

$$p = \exp\left(-\frac{E(r_{t+1}) - E(q_t)}{k_B T}\right). \tag{29}$$

5 Proposed system

This section describes the implementation of the proposed approach that is used to conduct the experiments in section 6.

Furthermore is the first research question ”How can Gaskett et al.’s method be adapted in order to work with computer vision on an embedded platform?” answered by describing the changes that were found to be necessary. The success of this is evaluated in section 6.

5.1 Hardware

The Raspberry Pi 2 Model B [RaspberryPiFoundation] was used to run the program on. The Raspberry Pi 2 Model B is a small ARM based computer that is developed by the Raspberry Pi foundation, which features a 900 MHz quad-core processor and 1 GB of ram. It was used with the Raspberry Pi camera module V1, in order to provide visual input to the system. It was powered with a 5000mhA USB battery back and connected over Wi-Fi with a USB WiFi dongle. The execution of actions was done with the educational robot Thymio II with a differential drive. The Thymio II [Prof Moti Ben-Ari] features different kinds of actuators and sensors which can be accessed over the USB interface. The hardware setup can be seen in Figure 17.

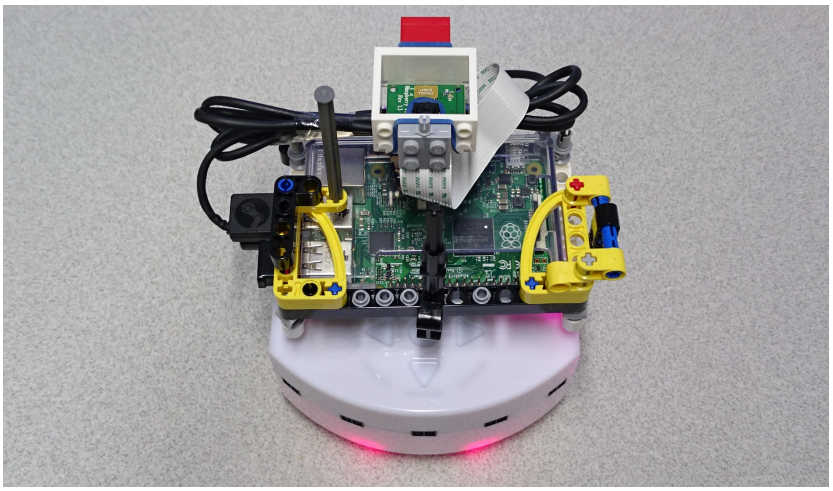

Figure 17: *The hardware setup.*

The choice of the embedded system and the robot platform is not that crucial since one of the goals of this project is that the developed system should be applicable on a variety of platforms. It is though worth noting that the Raspberry Pi 2 Model B is definitely not the fastest embedded platform

especially when it comes to performing parallel computations (essential for image computer vision). One of the reasons why the Raspberry Pi 2 Model B was chosen in this project, was that the computational limitations of the Raspberry Pi would ensure that the developed system also could run on less powerful embedded.

The other reason was that it runs a full Linux system, which offers a lot of flexibility in the development. One can freely decide what programming language to use and has more or less the same options as on a desktop computer.

The Raspberry Pi can be interacted with over a ssh connection together with a SFTP connection in order to have access to the files on the Raspberry Pi.

The Raspberry Pi can access the Thymio's actuators and sensors over the command-line utility asebamedulla. This can be done with third party languages such as Python, Perl or any language that supports the D-Bus protocol [Prof Moti Ben-Ari].

As this system should be vision based are only the actuators of the Thymio's used. The input of the system is taken from the Raspberry Pi camera module V1, which is accessed over the picamera library.

Reinforcement Learning System

In this section the reinforcement learning process that is implemented in this project is described. It is here focused on explaining the overall idea of the implementation, on the problems that were faced, and the way they were solved. All parts of the system were implemented in python in order to have full control over the system behavior and to be able to determine the state of all the variables at runtime.

The Framework

Since the implemented reinforcement learning techniques in this project are based on Q-learning, it is possible to describe them in a general framework.

A simplified version of this framework, can be seen in the flowchart in Figure 18. This flowchart shows the main loop of the system. The essential variables that have to be understood are the previous state of the system s, the current state s', the last taken action a and the reward r that was received due to action a.

When the system is started, there is no previous action or state, which is why they are both set to None. In the main loop, the state of the system is acquired base on the image from the camera. It is then checked if there is a previous state and action i.e. there is an experience to learn from. In this case the reward is calculated based on the new state and the model is adjusted in the learn function. The next step would be determining the next action that should be carried out based on the state s'. Before carrying out the action and

returning to the beginning of the loop again the state s' would be saved as the previous state s for the next iteration.

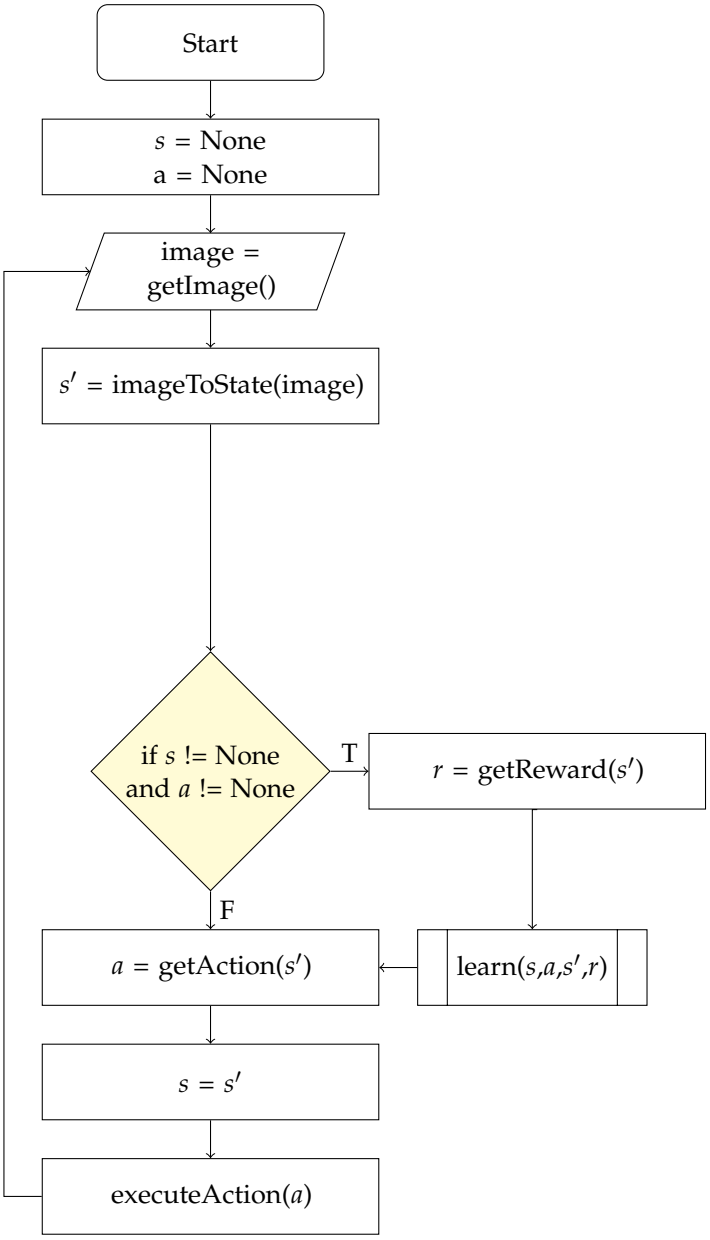

Figure 18: *The program flow of the reinforcement learning.*

Delayed State

The state of the system determined based on an acquired image from the camera. The execution of an action is done by setting the wheels of the robot to a specific speed with the dbus library [5].

The reinforcement learning therefore tries to learn the optimal policy for, which speed to set the motors of the differential drive to, dependent on the input from the camera in order to maximise its reward. An important thing to note is that acquiring the state of the system involves computationally expensive image processing. This means that the state of the system might not be the current state of the system any more when it is received by reinforcement learning. In fact, if the program shown in Figure 18, simply sets the wheel speed of the robot and immediately receives a new image from the camera, the action will have no effect on the following state, but on the state after that.

Furthermore, the effect of setting the wheels to a specific speed would depend on the time it takes to complete one program loop, which was measured not to be constant.

In order to avoid this problem, the system was setup to drive with a certain speed for a specific time, while pausing the program. The wheels were stopped before returning to the main loop. This leads to a less desirable start stop policy, but is necessary for a accurate state representation.

Learning Duration

Another time related problem is that the learn part seen in Figure 18 can take up very different amounts of time. This is because the learning time depends highly on the complexity of the model. Even with the same learning model, the time varies from run to run dependent on how much the model needs to be adjusted e.g. a neural network which is adjusted with back propagation until a certain error is reached. This either leads to a system that stalls or carries out the same action for a undefined long time, dependent on how the delayed state problem is handled.

In order to minimise this problem the learning is carried out in a separate thread. Every time a new experience is available, the system checks if the thread is idling in which case it will be assigned a new experience to learn from. This means that there might be experiences that are dropped, similar to frames that can't be processed fast enough. This is illustrated in Figure 19.

In order to avoid race conditions, two versions of the model used by the reinforcement learning exist. One is used for learning and the other to get actions for execution. After each learning step, the adjusted model is copied to replace the the executing model. This is done while stalling the other thread.

With this method the leaning process is reduced to the time it takes to copy the model.

[5] `https://www.freedesktop.org/wiki/Software/dbus/`

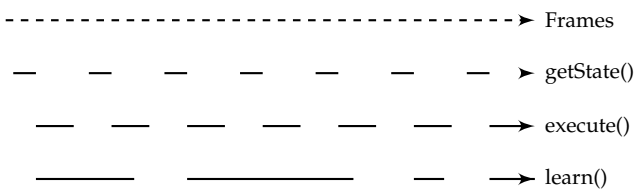

Figure 19: *The timing of the program.*

The Wire Fitter

The wire fitting interpolation was implemented as explained by Gaskett et al. [1999]. In order to make sure that it was being adapted correctly by the implemented gradient descent, was the value space plotted after making the system adapt to the same wires. One of these plots can be seen in Figure 20.

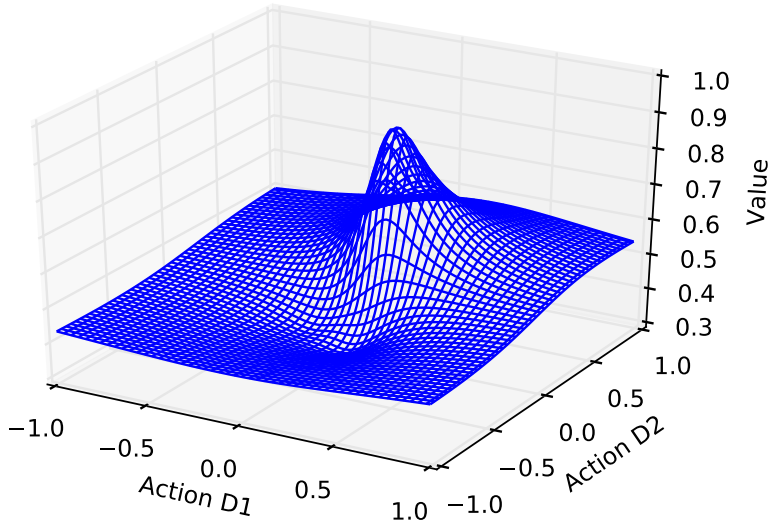

Figure 20: *A two dimensional action space*

5.2 Computer Vision State

With convolution neural networks being a very popular choice one could imagine to train a CNN to recognise landmarks in the environment that guide the navigation. Throwing problems like misclassified landmarks into the learning process seemed though a bit ambitious before having tested that the learning of the navigation actually worked. It was therefore chosen to use landmarks that are easy to recognise and to postpone the extension to CNNs.

Quick Response Code and unicoloured patches of cardboard were considered as easy to recognise landmarks.

As the classification of the unicoloured patches of cardboard was less susceptible to misclassification due to lens distortion and partially occlusion was it chosen over QR codes.

For the colour classification images of the colour patches were taken under different viewing conditions (angles and lighting). As the Pi camera module measures YUV420 values were these mapped to the HSV colour space. In the HSV colour space the mean and the variance of the colour were determined across the different samples.

As the hue is a circular value one needs to take extra care when determining its distribution. A reasonable measure of circular distance between two points A and B with angles $\theta 1$ and $\theta 2$ is given by eq. (30) from [Roy et al., 2012].

$$d(A, B) = 1 - cos(\theta_1 - \theta_2) \tag{30}$$

With the available mean and variance of the colour samples the Malahanobis distance between a new colour sample and the colour distribution can be determined. This can be used to measure if a new pixel belongs to the colour or not.

In order to make the classification as computational inexpensive as possible, a lookup table was computed. This lookup table directly maps the output from the camera to several class images. Each of the class images contains binary pixels describing if the colour is part of the class or not.

The final classification of 5 colours could be executed in $\tilde{2}$.74ms for an image with a QVGA resolution on the Raspberry Pi 2.

I order to determine the position of the colour patch in the image the centre of gravity in each class image was used.

6 Experiments

In this section the experiments that were conducted to answer the research questions are explained.

The question ”How well is the Q-function approximated by Gaskett et al.’s method?” is addressed in the Parameter test. As a comparison to Gaskett et al.’s method standard Q-learning is used on a rasterised state and action space. This test also serves the purpose of tuning the parameters of the learning method.

In the subsequent Non-simulated test the parameters are used. This test serves the purpose of showing that the proposed system for computer vision on an embedded platforms is working. It therefore confirms that the adaptation of Gaskett et al.’s system are working and helps to answer ”How can Gaskett et al.’s method be adapted in order to work with computer vision on an embedded platform?”

Furthermore the test is carried out in a non-simulated environment and can there be used to answer ”Is the adapted system working in a non-simulated environment?”.

6.1 The Adapted Task

In this section the task that is solved in the parameter test and the non-simulated test is explained. This task is held as close as possible to the task designed by Gaskett et al. while incorporating computer vision.

Gaskets Task

In the test conducted by Gaskett et al. the task was for a simulated submersible robot to reach 40 targets. The robot was given 200 time steps in order to reach each of the target. In other words, the target was changed every 200 time steps. The targets were placed randomly, but with a distance of one unit to each other. As a metric for how well the learning was performing, the average distance over each time period was recorded. This test was repeated 140 times. They showed that it is possible to reduce the average distance to the targets over time, i.e. the submersible robot reached the targets faster over time.

Computer Vision Task

The chosen task in this work is getting a robot with a differential drive to centre itself under a pattern on the ceiling. For this, the robot was equipped with a camera that was pointed orthogonal to the ceiling. The robot is therefore considered centred under the pattern when the pattern is located in the centre of the image acquired by the camera. The recognition of the pattern is not part of the learning algorithm and performed as described in section 5.2. The position of the centre of the pattern in the acquired image is therefore given directly to the robot. The relationship that has to be learned is how to use the differential drive to move the centre pixel of the pattern further to the centre of the image. An image of the test setup is shown in Figure 21, where the red dot on the ceiling is the pattern that the robot has to centre itself underneath.

Figure 21: *The test setup.*

The state of the system is described by a 2D vector that consists of the pixel position of the pattern in the acquired image. The state vector s can be seen in eq. (31), where x_p and y_p are the x any y of the centre of the pattern in the image. For the experiments that are done with Gaskett et al.'s method, the

state space variables are normalised to values between 0 and 1.

$$s = [x_p, y_p] \tag{31}$$

The actions that the system can take are also described in a 2D vector, which consists of the right r and of the left l wheel speed, as can be seen in eq. (32). These parameters are normalised to the maximum and forward and backward wheel speed and are ranging from full backward -1 to full forward +1.

$$a = [l, r] \tag{32}$$

The reward that is given to the systems is the negative distance between the centre of the image and the centre of the pattern in the image. This can be seen in eq. (33), where (x_p, y_p) is the centre of the pattern and (x_c, y_c) is the centre of the image.

$$r = \sqrt{(x_p - x_c)^2 + (y_p - y_c)^2} \tag{33}$$

6.2 The Parameteter Test Method

A common method to test reinforcement learning algorithms is to observe how they increase the reward that they get over time. This requires the system to follow the policy that is being learned.

Part of the experiment is to access the right parameters for the reinforcement learning algorithm. There are a lot of parameters that can be used in different setting combinations. That is why observing the reward while following the policy by moving the robot physically is rather impractical. It was instead chosen to compare how well the system could approximate the value function with different parameter settings. One of the advantages in Q-learning is that the system does not have to follow the policy that it learns. Experiences in the form of state-action-state can therefore be recorded while following a random policy.

These can later be send to different systems as if they were being experienced. The physical movement has only to be executed once during the recording.

During this test, the different systems are presented sequentially with all training samples. After each training sample presentation, the system is used to predict the value for each action in a separate test dataset. Since the state after each of the actions is known, the correct value can be calculated. The average error that the system does on the training dataset is seen as the error of the system.

The error produced depends on how difficult it is to describe the test dataset with a model. To get an idea about the difficulty, the single value that describes each test dataset best is calculated. This value is the best guess of the value that one could take when not knowing the state of the system. When the estimated

value of a system produces a lower error than this value, one can conclude that the system is not just guessing.

In order to find a good parameter combination most of the parameters were set to standard values. To find a better parameter combination the parameters were altered with simulated annealing.

The quality metric that was used for the simulated annealing was the error made during the presentation of the last 10 percent of the test samples.

In this experiment, 2000 samples of state-action-state triplets were recorded. These were recorded while the robot was following a fully random policy. The samples were split up into a test and a training dataset, contain 10 and 90 percent of samples, respectively.

The experiment was repeated 30 times in which the training and test dataset were re-sampled randomly from the collected samples. The mean and variance of the error across these experiments were captured to access the quality of the parameter setting.

The parameters that where optimised with simulated annealing for the system that uses standard Q-learning with a rasterised the state and the action space are:

- Raster size for the state space of states.
- Raster size for the action space of states.
- Learning rate for the Q-learning.

The parameters that were optimised for Gaskett et al.'s system were:

- The number of neurons in each layer of the neural network.
- The training rate for the neural network.
- The momentum for the neural network.
- The number of back propagation iterations that the neural network runs for every sample.
- The number of wires that the neural network has to output.
- The training rate for the wire fitting.
- The number of gradient descent iterations that the wire fitting process is runs for every sample.
- The training rate for the Q-learning update.
- The number of persistence excitation.

The variables that were held constant during this process were:

- The discount factor of the Q-learning
- The amount of advantage in the Q-learning

This was done since it would have a direct influence on the prediction of the Q-function. A system that uses Q-learning with a low discount factor will be better at predicting the Q-function, but worse at reaching long term goals.

6.3 The Non-simulated Test Method

In this test the parameters found in the parameter test are used. The experiment that was conducted, involved following the policy by moving the robot physically and recording how it improved the reward it gets. This was done with different parameter settings, that could not be tested in the parameter test. These are parameters that alter how the action state space is to be explored.

For the test, the robot was placed on a marked position with a known distance to the pattern that it had to centre itself under. Subsequently it was given 100 time steps to centre itself under the pattern. It was then moved back to the starting position facing a random direction. This was done, by prompting the tester to reset the system to one out of eight predefined orientations. The system would then get another 100 time steps to centre itself. This procedure was repeated in order to see if the system would get better at reaching the target position.

In the work from Gaskett et al. [1999], no exploration is mentioned. It was therefore tried to run the system with no exploration at all. This was both done with Q-learning and advantage learning. Further, two different exploration functions were tested as well as the effect of doing persistence excitation with randomised samples from the past and the most reason samples from the past.

6.4 Results

In this section the results of the tests are presented.

Parameter Tuning Results

The graph in Figure 22 shows the error made on the test dataset after every training sample. The blue curve is error curve for the best parameter combination from Gaskett et al.'s system and the green curve shows the best with a rasterised state and action space system. The area around the curves show the variance across the 30 conducted test while the lines show the mean value. The red line is the line that needs to be crossed in order to conclude that the system is not guessing.

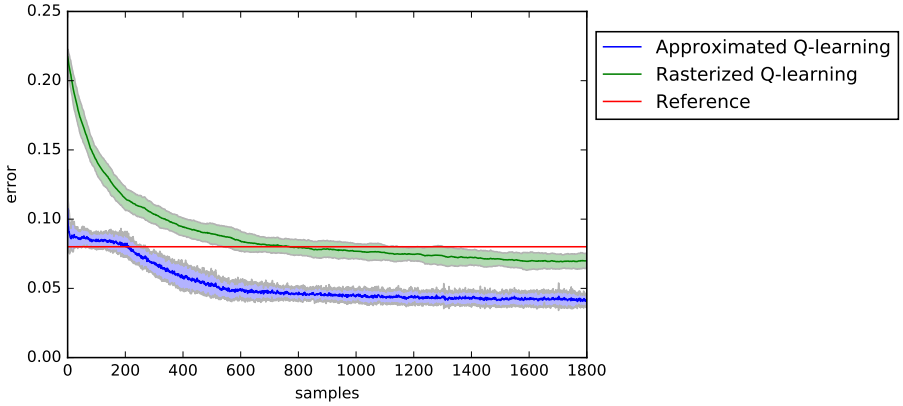

Figure 22: *The error curve for the best parameter combination for Gaskett et al.'s system and the system that uses standard Q-learning with a rasterised state and action space*

From Figure 22 it can be seen that both systems are getting better at predicting the outcome of actions over time. Gaskett et al.'s system crosses the line of guessing after 200 time steps, while it takes 800 time steps for the rasterised standard Q-learning. It can be seen that Gaskett et al.'s system is better than the the rasterised standard Q-learning at all time instances.

Non-simulated Results

The performance of the final of the system was measured by the distance to the target at every 100th time step in the non-simulated test. The distance from the system's target can be seen in Figure 23, where every line shows one system. The distance to the target at every starting position is 0.25. This distance is normalised to the size of the image. A distance shorter than this would therefore mean that the system got closer to the target over the 100 frames that it got.

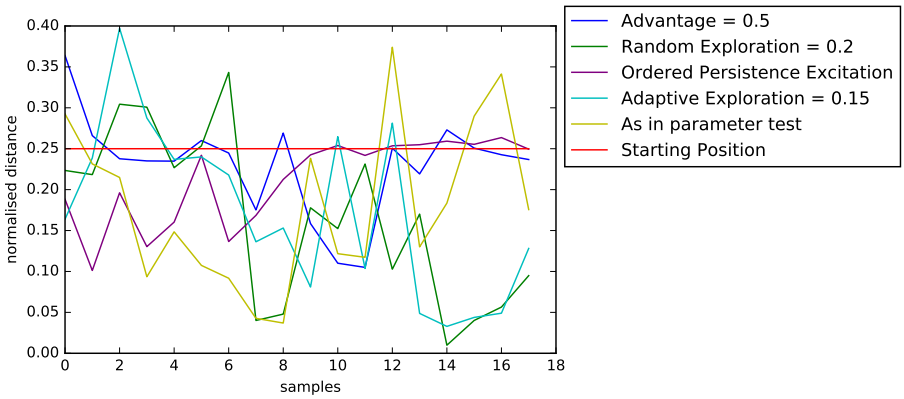

Figure 23: *The distance to the target after every 100 frames for different systems*

From Figure 23 can be seen that most of the test runs moved the robot closer to its target. By observing the systems it could be seen that the systems without exploration signal often got stuck in their policy. This included turning on the spot when the system was started and moving in circles towards the target. This problem is also reflected in the graph. Both the systems that used some form of exploration got closer to the target in the last runs, while the other systems got stuck with suboptimal policies.

It can be seen that the system that utilised advantage learning also got better at solving the task around the 800th sample, but then unlearned the policy again.

The problem with greedy algorithms that only choose the action that yields the highest result is that they are likely to get stuck in a local minima. In the case of the systems that got stuck this means that every action that moves the system away from the starting point yields a worse value than staying there.

Even though the system with the probability of 0.2 for a random action seem to work well in Figure 23 will it keep doing action no mater how sure it is of the situation. The adaptive exploration is therefore more elegant for some cases where the probability for exploration should be bound to the certainty of the model.

The probability for random actions during the test was recorded and is shown in Figure 24.

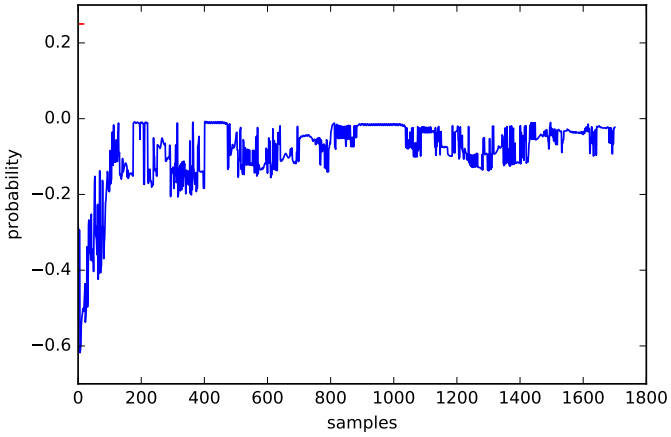

Figure 24: *The probability for random action in the adaptive exploration*

7 Conclusion

In this project it is shown that it is possible to design a system, that utilises reinforcement learning for navigation in a non-simulated environment. Several systems were run in in a non-simulated environment and were able to improve their performance over time.

In the first experiment that was conducted was it possible to show that the system designed by Gaskett et al. [1999] was able to predict the Q-value for a test dataset. Furthermore it was possible to show that this method was better than using standard Q-learning on rasterised states and actions.

The final solution is working in a non-simulated environment and is using computer vision on an embedded platform. Which answers the research question raised in this work.

During the development of the prototype has it become clear that the setting of the parameters for a reinforcement learning system is not trivial. By first finding a parameter combination that worked for predicting the Q-value, and later improving this setting with simulated annealing it was possible to find a combination of parameters that worked for the task solved in this project.

8 Future Development

Since it was possible to design a system that could centre itself under a patter would it be interesting to test how the system would behave for more complex scenarios. This could include, using the system in a hierarchy learning task as

it was suggested by Bischoff et al. or by using it in a task that requires a higher dimensional state and actions space.

Another way the project could be extended is by conducting a test that compares the performance of Gaskett et al.'s reinforcement method against Stanley and Miikkulainen's NEAT.

Furthermore would it be of interest to see if the performance of the system in the described task could be improved. This could be done by further improving the systems parameter setting.

One of the observations that was made in the physical test was that even though the robot reached its destination it would still turn on the spot. It would therefore be a natural next step to include a penalty for movement. This could lead to that the robot would move more directly to its target position and learn to stand still, when the target position is reached.

9 Acknowledgement

This project was written in collaboration with Movidius LtD. I would therefore like to thank the entire team for taking me in as an intern and supporting me through the entire project.

References

Astuti, Giudice, Longo, Melita, Muscato, and Orlando, 2009. G. Astuti, G. Giudice, D. Longo, C. D. Melita, G. Muscato, and A. Orlando. *An Overview of the "Volcan Project": An UAS for Exploration of Volcanic Environments*. pages 471–494, 2009. doi: 10.1007/978-1-4020-9137-7_25. URL `http://dx.doi.org/10.1007/978-1-4020-9137-7_25`.

Barry, Brick, Connor, Donohoe, Moloney, Richmond, O'Riordan, and Toma, Mar 2015. B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma. *Always-on Vision Processing Unit for Mobile Applications*. IEEE Micro, 35(2), 56–66, 2015. ISSN 0272-1732. doi: 10.1109/MM.2015.10.

Basch, Cristea, Tiponuţ, and Slavici, 2010. Mihai-Emanuel Basch, David-George Cristea, Virgil Tiponuţ, and Titus Slavici. *Elaborated Motion Detector Based on Hassenstein-Reichardt Correlator Model*. pages 192–195, 2010. URL `http://dl.acm.org/citation.cfm?id=1984140.1984181`.

Bellman, 1957. Richard Bellman. *A Markovian Decision Process*. Indiana Univ. Math. J., 6, 679–684, 1957. ISSN 0022-2518.

Bischoff, Nguyen-tuong, Lee, Streichert, and Knoll, 2013. B. Bischoff, D. Nguyen-tuong, I h. Lee, F. Streichert, and A. Knoll. *Hierarchi-*

cal Reinforcement Learning for Robot Navigation. 21st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning, 2013. URL http://www6.in.tum.de/Main/Publications/BischoffESANN13a.pdf.

Bonin-Font, Ortiz, and Oliver, 2008. Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. *Visual Navigation for Mobile Robots: A Survey*. Journal of Intelligent and Robotic Systems, 53(3), 263–296, 2008. ISSN 1573-0409. doi: 10.1007/s10846-008-9235-4. URL http://dx.doi.org/10.1007/s10846-008-9235-4.

Chatila, Lacroix, Simeon, and Herrb, 1995. Raja Chatila, Simeon Lacroix, Thierry Simeon, and Matthieu Herrb. *Planetary exploration by a mobile robot: Mission teleprogramming and autonomous navigation*. Autonomous Robots, 2(4), 333–344, 1995. ISSN 1573-7527. doi: 10.1007/BF00710798. URL http://dx.doi.org/10.1007/BF00710798.

Cherni, Boutereaa, Rekik, and Derbel, Oct 2015. F. Cherni, Y. Boutereaa, C. Rekik, and N. Derbel. *Autonomous mobile robot navigation algorithm for planning collision-free path designed in dynamic environments*. pages 1–6, 2015. doi: 10.1109/JIEEEC.2015.7470747.

Cope A.J., Rationalchmond P. and D, 2013. Marshall J. Cope A.J., Rationalchmond P. and Allerton D. *Creating and Simulating Neural Networks in the Honeybee Brain using a Graphical Toolchain*. 2013. URL http://greenbrain.group.shef.ac.uk/wp-content/uploads/2013/11/SFN_2013_GB.pdf. Poster presented at the Society for Neuroscience Annual Meeting, Nov. 2013, University of Sheffield, San Diego.

D'Addona, 2014. Doriana Marilena D'Addona. *CIRP Encyclopedia of Production Engineering*. pages 911–918, 2014. doi: 10.1007/978-3-642-20617-7_6563. URL http://dx.doi.org/10.1007/978-3-642-20617-7_6563.

Democrat and Chronicle, 2016. Democrat and Chronicle. *Domestic drone accidents*. 2016. [Online; accessed 16-May-2016].

Dini and Serrano, 2012. Steve Dini and Mark Serrano. *Combining Q-Learning with Artificial Neural Networks in an Adaptive Light Seeking Robot*. 2012. URL http://web.cs.swarthmore.edu/~meeden/cs81/s12/papers/MarkStevePaper.pdf.

Dotenco, Gallwitz, and Angelopoulou, 2015. Sergiu Dotenco, Florian Gallwitz, and Elli Angelopoulou. *Autonomous Approach and Landing for a Low-Cost Quadrotor Using Monocular Cameras*. pages 209–222, 2015. doi: 10.1007/978-3-319-16178-5_14. URL http://dx.doi.org/10.1007/978-3-319-16178-5_14.

Drchal, Koutnik, and Snorek, **2009**. Jan Drchal, Jan Koutnik, and Miroslav Snorek. *HyperNEAT Controlled Robots Learn How to Drive on Roads in Simulated Environment*. page 6, 2009.

Dresscher, Coelen, Broenink, and Stramigioli, **2014**. Douwe Dresscher, Michiel van der Coelen, Jan Broenink, and Stefano Stramigioli. *Control and Omni-directional Locomotion of a Crawling Quadruped*. pages 486–497, 2014. doi: 10.1007/978-3-319-11900-7_41. URL http://dx.doi.org/10.1007/978-3-319-11900-7_41.

Emigh, Kriminger, and Principe, **Sept 2015**. M. Emigh, E. Kriminger, and J. C. Principe. *A model based approach to exploration of continuous-state MDPs using Divergence-to-Go*. pages 1–6, 2015. ISSN 1551-2541. doi: 10.1109/MLSP.2015.7324371.

Gaskett, Wettergreen, and Zelinsky, **1999**. Chris Gaskett, David Wettergreen, and Alexander Zelinsky. *Advanced Topics in Artificial Intelligence: 12th Australian Joint Conference on Artificial Intelligence, AI'99 Sydney, Australia, December 6–10, 1999 Proceedings*. pages 417–428, 1999. doi: 10.1007/3-540-46695-9_35. URL http://dx.doi.org/10.1007/3-540-46695-9_35.

Howard, **1960**. R.A. Howard. *Dynamic Programming and Markov Processes*. 1960. URL https://books.google.co.uk/books?id=fXJEAAAAIAAJ.

Jatsun, Loktionova, and Malchikov, **2014**. S. Jatsun, O. Loktionova, and A. Malchikov. *Six-Link In-pipe Crawling Robot*. pages 341–348, 2014. doi: 10.1007/978-3-319-07058-2_38. URL http://dx.doi.org/10.1007/978-3-319-07058-2_38.

Katzschmann, Marchese, and Rus, **2016**. Robert K. Katzschmann, Andrew D. Marchese, and Daniela Rus. *Hydraulic Autonomous Soft Robotic Fish for 3D Swimming*. pages 405–420, 2016. doi: 10.1007/978-3-319-23778-7_27. URL http://dx.doi.org/10.1007/978-3-319-23778-7_27.

Lee and Kim, **2013**. Seung-Jae Lee and Jong-Hwan Kim. *Development of a Quadrocoptor Robot with Vision and Ultrasonic Sensors for Distance Sensing and Mapping*. pages 477–484, 2013. doi: 10.1007/978-3-642-37374-9_46. URL http://dx.doi.org/10.1007/978-3-642-37374-9_46.

Manfredi, Assaf, Mintchev, Marrazza, Capantini, Orofino, Ascari, Grillner, Wallén, Ekeberg, Stefanini, and Dario, **2013**. L. Manfredi, T. Assaf, S. Mintchev, S. Marrazza, L. Capantini, S. Orofino, L. Ascari, S. Grillner, P. Wallén, Ö. Ekeberg, C. Stefanini, and P. Dario. *A bioinspired autonomous swimming robot as a tool for studying goal-directed locomotion*. Biological Cybernetics, 107(5), 513–527, 2013. ISSN 1432-0770. doi: 10.1007/s00422-013-0566-2. URL http://dx.doi.org/10.1007/s00422-013-0566-2.

McNabb, **2016**. Miriam McNabb. *Changing Forecasts: The Drone Industry Surprise*. 2016. [Online; accessed 16-May-2016].

Nagai, Mizushina, Nakamura, Sugimoto, Watari, Nakajo, and Yoshida, **2015**. Mamoru Nagai, Asuka Mizushina, Taro Nakamura, Fumitaka Sugimoto, Kensuke Watari, Hidehiko Nakajo, and Hiroshi Yoshida. *Development of a Hydraulic Artificial Muscle for a Deep-Seafloor Excavation Robot with a Peristaltic Crawling Mechanism*. pages 379–389, 2015. doi: 10.1007/978-3-319-22879-2_35. URL http://dx.doi.org/10.1007/978-3-319-22879-2_35.

Nikolaev and Iba, **2006**. Nikolay Y. Nikolaev and Hitoshi Iba. *Adaptive Learning of Polynomial Networks: Genetic Programming, Backpropagation and Bayesian Methods*. pages 147–180, 2006. doi: 10.1007/0-387-31240-4_6. URL http://dx.doi.org/10.1007/0-387-31240-4_6.

Pardalos and Mavridou, **2009**. Panos M. Pardalos and Thelma D. Mavridou. *Encyclopedia of Optimization*. pages 3591–3593, 2009. doi: 10.1007/978-0-387-74759-0_617. URL http://dx.doi.org/10.1007/978-0-387-74759-0_617.

Prof Moti Ben-Ari, Christophe Barraud. Dr Stéphane Magnenat Morgane Chevalier Dr Gordana Gerber Manon Briod Maria Beltran Prof Moti Ben-Ari, Christophe Barraud. *Thymio*. URL https://www.thymio.org/. Downloaded: 01/06-2016.

RaspberryPiFoundation. RaspberryPiFoundation. *RASPBERRY PI 2 MODEL B*. URL https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Downloaded: 01/06-2016.

Roy, Parui, and Roy, **November 2012**. Anandarup Roy, Swapan K. Parui, and Utpal Roy. *Article: A Mixture Model of Circular-Linear Distributions for Color Image Segmentation*. International Journal of Computer Applications, 58(9), 6–11, 2012. Full text available.

Stanley and Miikkulainen, **2002**. Kenneth O. Stanley and Risto Miikkulainen. *Evolving Neural Networks Through Augmenting Topologies*. Evolutionary Computation, 10(2), 99–127, 2002. URL http://nn.cs.utexas.edu/?stanley:ec02.

Stone, **2014**. Peter Stone. *Encyclopedia of Machine Learning and Data Mining*. pages 1–1, 2014. doi: 10.1007/978-1-4899-7502-7_689-1. URL http://dx.doi.org/10.1007/978-1-4899-7502-7_689-1.

Suárez, Heredia, and Ollero, **2014**. Alejandro Suárez, Guillermo Heredia, and Aníbal Ollero. *Analysis of Perturbations in Trajectory Control Using Visual Estimation in Multiple Quadrotor Systems*. pages 115–129, 2014. doi: 10.1007/978-3-319-03413-3_9. URL http://dx.doi.org/10.1007/978-3-319-03413-3_9.

Sutton and Barto, **2005**. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, London, England, 2005. [Online; accessed 16-May-2005].

Tan and Cai, **Dec 2015**. P. Tan and Z. Cai. *Modelling and Planning of Mobile Robot Navigation Control in Unknown Environment*. pages 1532–1536, 2015. doi: 10.1109/CICN.2015.292.

Tesen, Saga, Satoh, and Nagase, **2013**. Satoshi Tesen, Norihiko Saga, Toshiyuki Satoh, and Jun-ya Nagase. *Peristaltic Crawling Robot for Use on the Ground and in Plumbing Pipes*. pages 267–274, 2013. doi: 10.1007/978-3-7091-1379-0_33. URL `http://dx.doi.org/10.1007/978-3-7091-1379-0_33`.

Tesla, **2016**. Tesla. *A Tragic Loss*. 2016. [Online; accessed 10-Sep-2016].

Velez, **2009**. Roby Velez. *Using NEAT to teach a crawling robot to follow a light*. 2009. URL `https://www.cs.swarthmore.edu/~meeden/cs81/s09/finals/Roby.pdf`.

Vieira, Silva, and Ferreira, **2016**. Micael T. L. Vieira, Manuel F. Silva, and Fernando J. Ferreira. *Design and Development of a Biological Inspired Flying Robot*. pages 231–243, 2016. doi: 10.1007/978-3-319-27146-0_18. URL `http://dx.doi.org/10.1007/978-3-319-27146-0_18`.

Vreeken, **2003**. Jilles Vreeken. *Spiking Neural Networks, an Introduction*. 2003.

Watkins and Dayan, **1992**. Christopher J. C. H. Watkins and Peter Dayan. *Q-learning*. Machine Learning, 8(3), 279–292, 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `http://dx.doi.org/10.1007/BF00992698`.

Yuan, Yu, Wu, and Tan, **2016**. Jun Yuan, Junzhi Yu, Zhengxing Wu, and Min Tan. *Precise planar motion measurement of a swimming multi-joint robotic fish*. Science China Information Sciences, 59(9), 1–15, 2016. ISSN 1869-1919. doi: 10.1007/s11432-015-5497-1. URL `http://dx.doi.org/10.1007/s11432-015-5497-1`.

Yusof, Mansor, and Baba, **Dec 2015**. Y. Yusof, H. M. A. H. Mansor, and H. M. D. Baba. *Simulation of mobile robot navigation utilizing reinforcement and unsupervised weightless neural network learning algorithm*. pages 123–128, 2015. doi: 10.1109/SCORED.2015.7449308.

Zhao and Chen, **2016**. Zhe Zhao and Xiaoping Chen. *Building 3D semantic maps for mobile robots using RGB-D camera*. Intelligent Service Robotics, pages 1–13, 2016. ISSN 1861-2784. doi: 10.1007/s11370-016-0201-x. URL `http://dx.doi.org/10.1007/s11370-016-0201-x`.